CARDINAL-Vanilla: Immune System Inspired Prioritisation and Distribution of Security Information for Industrial Networks

Peter M. D. Scully

Department of Computer Science
Aberystwyth University
Wales

June
2016

This thesis is submitted in partial fulfilment of the
requirements for the degree of

Doctor of Philosophy of Aberystwyth University.

Declaration

This thesis has not previously been accepted in substance
for any degree and is not being concurrently submitted in candidature
for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except
where otherwise stated.

Other sources are acknowledged by footnotes giving explicit
references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give consent for my thesis, if accepted, to be made
available for photocopying and for inter-library loan, and for the
title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Acknowledgements

The opportunity to study at the level of philosophical doctorate, under such pleasant study conditions leaves me sincerely in a debt of thanks. This is a debt that, truly, I hope many many others will have the good fortune to carry. To have such freedom and time to educate yourself and to learn the skills to critically evaluate what you see before you are so vital and so absolutely needed by individuals from all walks of life.

To Tan, sharing these life experiences with you over each year has made them infinitely better. And my wonderful growing family, for your unfaltering support, unquestioned confidence and firm positive motivation, I thank you sincerely.

To Mark J. Neal, my PhD supervisor, I have much respect and thanks for you. Your insight, lateral-thinking, criticisms and critiques have necessarily guided me into and through many a tough academic and applied time over the past 4 years. With your clear vision and expertise, your commonplace *ahoy!-like* jovial cheer, your patience and your commitment throughout the trials-and-failures, you have made this experience all the more simpler and enjoyable.

To Jules Ferdinand Pagna Disso, my industrial PhD supervisor, your direction, support, prompting and prodding along the path has been so helpful. The technical training and academic insight you have given has let us to bring these very different worlds of study together.

To David E. Price, my technical PhD supervisor, I gratefully thank your critiques and your thirst for and requests of added detail. Of course, the hours you have set aside and your area-specific know-how in distributed and networked systems made problem identification and development much much smoother.

Thanks Marek Ososinski for all you have done. Thanks to Tom Blanchard and Colin Sauzé for our numerous insightful discussions through the possibilities of bio-metaphors, and for your understanding. Thanks to Jingping Song and Viktoriya Degeler for our discussions and collaborations.

There are many other friends and colleagues who have discussed and made contributions to the decisions, priorities and approaches in this work, each have helped along the way, you know who you are and I thank you kindly.

Of course this study would not have been possible without the financial sponsorship from the EPSRC industrial CASE studentship and grant EP/J501785/1 and from the EADS Innovation Works research program IW201339 grant.

Contents

1 Introduction **11**

1.1 Motivations and Context . . . 11
1.2 SHAAM PhD Project . . . 12
1.3 Aims . . . 13
1.4 Objectives . . . 13
1.5 Hypothesis and Research Question . . . 14
1.6 Key Contributions . . . 14
1.7 Key Testing Areas in this Thesis . . . 15
1.7.1 Distribution of Security Modules and Benchmarking . . . 15
1.7.2 Architecture Parameter Tuning . . . 15
1.8 Thesis Scope and Organisation . . . 16

2 Security in Industrial Networks **19**

2.1 Introduction . . . 19
2.2 Standards . . . 20
2.3 Latest Vulnerabilities & Exploits . . . 20
2.4 Historical Issues in the Security of Industrial Networks . . . 24
2.5 Attacker Opportunities and Evidence . . . 24
2.6 Academic Summaries of Automation Security Flaws . . . 26
2.7 The Case and Positioning for Automated Self-Healing Security Systems in Industrial Networks . . . 26
2.8 Problems . . . 27
2.9 Main Challenges and Directed Focus . . . 28

3 Biological Self-Healing **29**

3.1 Introduction . . . 29
3.2 The Innate Immune System . . . 30
3.2.1 Tissue Immune Defences . . . 30
3.2.2 Neutrophils Response . . . 31
3.2.3 Complement System Responses . . . 31
3.3 The Adaptive Immune System . . . 32
3.3.1 B-Cells and Adapting Antigenic Specificity . . . 32
3.3.2 T-Cells: Dealing with Viruses, Regulation and Homeostasis . . . 33

3.3.3 Antigen Presenting Cells . . . 35
3.4 Further Homeostatic Behaviours . . . 36
3.4.1 Signalling . . . 36
3.4.2 Two Signal Theory: Co-stimulation, Co-receptors and Binding Thresholds . . . 37
3.4.3 Evolving Self-Healing . . . 37
3.4.4 Cell Potency: Structural Adaptation and Organisation . . . 38
3.5 Chapter Conclusions . . . 40
3.5.1 Mappings of Principles to Main Challenges . . . 40

4 Artificial Self-Healing **43**
4.1 Introduction to Self-Healing Systems . . . 43
4.2 A Survey of Distributed Self-Healing Artificial Immune Systems for Network Security . 44
4.2.1 Early Distributed Intrusion Detection Systems (DIDS) . . . 45
4.2.2 File Decoys and Negative Selection in DIDS . . . 45
4.2.3 File Decoys, Negative Selection and Grid Computing for Evaluation . . . 46
4.2.4 Backup and Restore, Block Transmissions, Neuter Viruses . . . 46
4.2.5 Negative Selection and Decentralised Lisys . . . 48
4.2.6 Toward an Holistic Self/Non-Self-inspired Defence System . . . 48
4.2.7 Toward an Holistic Danger Theory-based Defence System . . . 50
4.2.8 A Sting for Worms with Self-Hardening . . . 51
4.2.9 Danger Theory Modelled for DIDS . . . 53
4.2.10 Cytokine Communications in Decentralised Defense Systems . . . 54
4.2.11 Intrusion Prevention and Multi-Agent Self-Healing . . . 54
4.3 Chapter Conclusions . . . 55
4.3.1 Discussion . . . 56
4.4 The Road Ahead . . . 56

5 The CARDINAL-Vanilla Architecture **59**
5.1 Introduction . . . 59
5.2 Clarifications and Differences . . . 60
5.3 Architecture Overview . . . 61
5.3.1 Engineered System Overview . . . 61
5.3.2 Immune System Inspiration Overview . . . 63
5.3.3 Agent Architecture Overview . . . 63
5.4 Processes of CARDINAL-Vanilla . . . 64
5.4.1 Process Differences . . . 66
5.5 Flow Control of CARDINAL-Vanilla . . . 67
5.5.1 Flow Control Differences . . . 67
5.6 Network Communications . . . 67
5.6.1 Network Connectivity & Decision Control . . . 69
5.6.2 Network Protocol . . . 69
5.7 Classification and Responses . . . 69

5.7.1 Primary Classifier . . . 70
5.7.2 Secondary Classifier . . . 71
5.7.3 Responses . . . 71
5.8 Module Creation & Validation . . . 71
5.8.1 Module Validation Process . . . 72
5.8.2 Module Migration to Tissue Process . . . 72
5.8.3 Choosing Thresholds and Cytokine Increase Values . . . 73
5.9 Time Scale: Moving Time Window . . . 73
5.10 Module Dispatch Decisions . . . 74
5.10.1 Volume Selection Feedback Loop . . . 74
5.10.2 Destination Selection . . . 75
5.10.3 Module Selection for Transmission . . . 75
5.11 Priority Heuristic for Signature Module Distribution . . . 75
5.11.1 Module Prioritisation & De-Prioritisation . . . 76
5.11.2 More Prioritisation: Through Regulation . . . 77
5.11.3 More Prioritisation: Dendritic Cells and Decay Rates . . . 78
5.12 Parameters of CARDINAL-Vanilla . . . 79
5.12.1 Choosing the Key Parameters . . . 80
5.12.2 Under Attack Volume Percentage Parameter (P0) . . . 80
5.12.3 Initial Priority Value Multiplier Parameter (P1) . . . 81
5.12.4 Priority Value Suppression Parameter (P2) . . . 81
5.12.5 Static Moving Window Size Parameter (P3) . . . 81
5.12.6 Dendritic Cell Lifespan Parameter (P4) . . . 82
5.13 Chapter Conclusions . . . 82
5.13.1 Next Steps . . . 83

6 How to Evaluate and Validate Distributed Self-Healing Security Systems 87
6.1 Introduction . . . 87
6.2 Measurements and Metrics . . . 87
6.2.1 Measures to Metrics . . . 88
6.2.2 Detector Distribution Quantity Measurement (M1) . . . 88
6.2.3 Detector Distribution Time Measurement (M2) . . . 89
6.2.4 Detector Distribution Data Sent Measurement (M3) . . . 90
6.2.5 Sources of Noise affecting Measurements . . . 90
6.3 Experiment Phases . . . 91
6.4 Experiment Procedure . . . 92
6.4.2 Experiment Procedure Development . . . 92
6.4.1 Experiment Procedure J . . . 93
6.5 Experiment Constants . . . 94
6.6 Distributed System User Behavioural Model . . . 95
6.6.1 Three Parameter User Model . . . 95

6.6.2 Static & Runtime Model Definition . . . 95
6.6.3 Distinguishing User Learning Experiences from User Behaviour . . . 97
6.6.4 Model's Use of Random Number Generators . . . 98
6.7 Datasets . . . 101
6.7.1 CSIC HTTP 2010 Dataset Versions and Sampling . . . 101
6.8 Real-time vs Post Processing . . . 104
6.9 Multi-Objective Evaluation Metrics . . . 104
6.10 Combined System Performance Measures . . . 105
6.10.1 Approach . . . 105
6.10.2 Rodriguez & Weisbin's Equations . . . 106
6.10.3 Ratio of Distances . . . 106
6.10.4 Conclusion for Combined System Performance Measures . . . 109
6.11 Chapter Conclusion . . . 110
6.11.1 Future Work upon the User Model . . . 110
6.11.2 Next Steps . . . 110

7 Self-Healing Benchmark **111**
7.1 Introduction . . . 111
7.2 Comparison on Virtual Networks . . . 112
7.2.1 Objectives . . . 112
7.2.2 Algorithm Comparisons . . . 112
7.2.3 Experiment Design . . . 114
7.2.4 Configuration . . . 116
7.3 Results . . . 118
7.3.1 Metric Performance . . . 118
7.3.2 Immunisation Rate System Performance . . . 120
7.3.3 Further Discussion . . . 124
7.3.4 Conclusions . . . 126
7.3.5 Next Steps . . . 127
7.4 Comparison on Enterprise Networks . . . 128
7.4.1 Objectives . . . 128
7.4.2 Experiment Design . . . 128
7.4.3 Configuration . . . 129
7.5 Results . . . 131
7.5.1 Metric Performance . . . 131
7.5.2 Immunisation Rate System Performance . . . 137
7.5.3 Conclusions . . . 138
7.5.4 Next Steps . . . 140
7.5.5 Future Work . . . 140
7.6 Further Analysis of Network Testing . . . 141
7.6.1 Objectives . . . 141

7.6.2 Observations of Time Synch/ Desynch in Network Virtualisation 142
7.6.3 Effects of Time Desynchronisation 145
7.6.4 Metric Differences with Time Desynchronisation 145
7.6.5 Detector Delivery Time Differences with Time Desynchronisation 148
7.6.6 Discussion 151
7.6.7 Conclusion 152
7.6.8 Future Work 152
7.7 Chapter Conclusion 153
7.7.1 Next Steps 153
7.7.2 Future Work 153

8 Parameter Tuning **155**
8.1 Introduction 155
8.2 Range Testing on Virtual Networks 156
8.2.1 Objectives 156
8.2.2 Experiment Design 156
8.3 Results 158
8.3.1 Part 1: Parameter Effects 158
8.3.2 Discussion of Results - Part 1: Parameter Effects 163
8.3.3 Part 2: Best Immunisation Rate Performance 164
8.3.4 Discussion of Results - Part 2: Best Immunisation Rate 166
8.3.5 Conclusions 168
8.3.6 Next Steps 168
8.4 Parameter Tuning on Virtual Networks 169
8.4.1 Objectives 169
8.4.2 Experiment Design 169
8.4.3 Search Method 169
8.4.4 Parameter Tuning Ranges 170
8.5 Results 170
8.5.1 Part 1: Search Results by Generation 171
8.5.2 Discussion - Part 1: Search Results by Generation 173
8.5.3 Part 2: Best Search States 173
8.5.4 Discussion - Part 2: Best Search States 175
8.5.5 Conclusions 175
8.5.6 Next Steps 177
8.6 Parameter Tuning on Enterprise Networks 178
8.6.1 Experiment Design 178
8.6.2 Parameter Tuning Ranges 178
8.6.3 Search Method 178
8.7 Results 180
8.7.1 Reasoning through the Noise 181

8.7.2 Best Configuration Set Results . . . 182
8.7.3 Discussion . . . 186
8.7.4 Conclusion . . . 187
8.8 Chapter Conclusions . . . 188
8.8.1 Peripheral Discoveries . . . 188
8.8.2 Future Work . . . 189

9 Towards a Decentralised Self-Healing Security System for Industrial Networks 191
9.1 Design Principles . . . 191
9.2 Self-Healing Component . . . 192
9.2.1 Role Separation . . . 193
9.2.2 Additional Hardware . . . 193
9.2.3 Performance Indicators . . . 194
9.2.4 Belief and Objective Weightings . . . 195
9.2.5 Periodic Indicator Updates . . . 195
9.2.6 Current Trust and Self-Centric View . . . 196
9.2.7 Fitness Function of Performance . . . 196
9.2.8 Data Transmission . . . 197
9.2.9 Detection Modelling . . . 197
9.2.10 Extracting Recovery State Models . . . 198
9.3 Self-Management Component . . . 199
9.3.1 Social Sensing and Collective Awareness – *"the guards themselves become the threat"* . . . 199
9.3.2 Moving Target Strategies – *"who will guard the guards themselves"* . . . 201
9.4 Strategy Generation and Evaluation: Static vs. Dynamic . . . 201
9.5 Application Focus . . . 202
9.6 Challenges . . . 202
9.7 Chapter Conclusions . . . 203

10 Conclusions 205
10.1 Contributions for Industrial Network Security . . . 206
10.2 Contributions for Artificial Immune Systems . . . 207
10.3 Summaries and Conclusions . . . 207

A Security in Industrial Networks 211
A.1 Industrial Security Standards . . . 211
A.1.1 Organisations with Released Industrial Security Standards . . . 211
A.1.2 Released Industrial Security Standards . . . 211
A.2 Vulnerabilities Reported on OSVDB 2007–2015 . . . 212
A.2.1 SCADA and Web Vulnerabilities Reported on OSVDB . . . 212
A.2.2 Vulnerabilities per PLC Manufacturer Reported on OSVDB . . . 214
A.3 SNAP7 - Open Source S7 Communications API . . . 215

A.4 Extract of ICS Attack - Water.arff Dataset 216

B Biological Immune System **217**

C CARDINAL-Vanilla Architecture **219**
C.1 Implementation Specific Responses 219
C.2 Use of Random Number Generators 220

D Configurations **223**
D.1 Virtual Machine Environment Configuration 223
D.2 Enterprise Machine Environment Configuration 223
D.3 Virtual Machine Environment and Execution Script 224
D.4 Enterprise Network Environment Parameter Search Execution Script 224
D.4.1 Time Evaluations of Other Datasets 226

E Further Results and Analysis **227**
E.1 Self-Healing Benchmark - Inferential Statistics 227
E.1.1 Difference from AIS to Engineered 227
E.2 Self-Healing Benchmark - Virtual Network Plot Results 229
E.3 Self-Healing Benchmark - Enterprise Plot Results 232
E.4 Time Delay: Bash Background Process Execution Order 235

F Parameter Range Evaluations **237**
F.1 Combined System Performance Measures: Part 2 237
F.1.1 Failed Route: Log Ratio with Inverse 237
F.2 Descriptive Statistics of Each Parameter Value Range 240
F.3 Box Plots of Each Parameter Value Range 240
F.4 Ordinal Correlation from Parameter Value to Metric Results 240
F.5 Descriptive Statistics of Immunisation Rates 247
F.6 Box Plots of Immunisation Rates 247
F.7 Best of Search States from Virtual Network Environment Tests 247
F.8 Enterprise Network Environment Parameter Tuning Tests 252
F.8.1 Initial Configuration Set Random Selection 252
F.8.2 Other Result Sets 252

Glossary **257**

References **269**

Chapter 1

Introduction

1.1 Motivations and Context

In a world of ever-changing self-sustaining malicious infections and ever-changing, complex, highly-connected computer systems an effective and equally self-adaptive countermeasure will be a vital tool. Like the components of the human immune system, these computational decision systems will need heterogeneous self-organisation, cooperative and autonomous system management with repairable failure-points, collaborative decision making, trusted and verifiable interactions and will need to evolve self-healing behaviours to sustain the dependable operational state and the fluctuating homeostasis of our cyber and computer networks.

The day of autonomous and intelligent malicious software is near. Within the next 20 years today's million-node botnets will have advanced evasive and polymorphic behaviours at all levels of implementation, they will supportively collaborate and maintain their existence with dedicated and automatic investigation of new vulnerabilities, system fingerprinting and propagation via replication. Their evolving objectives will guide their actions toward dominance over individuals, organisations and software entities that threaten them. They will learn and evaluate new mechanisms to cause impact, finding new ways to take control of newly developed and connected robotic technologies. These, among other fictional possibilities, may be the news headlines of the future.

Today's headlines in security-centric communities hail of cyberwar, cyberterrorism and countless intellectual property thefts. In 2010, a number of large U.S. owned companies, including Google and Yahoo, have been a target of motivated attacks with advanced persistent threats (APT) (McClure *et al.* , 2010). In 2013, we saw more reports of state-on-state incursions for corporate intellectual property theft with resident APTs (Mandia, 2013). However, it is Stuxnet's audacity that still resonates in halls of cyber security; a targeted attack on Iran's critical infrastructure that caused physical damage, yet is believed to have remained hidden for a number of months (Cherry, 2010).

Works in the academic research field of Artificial Immune Systems (AIS) have highlighted interesting cognitive mechanisms that can be applied to address the challenges we face. There are intriguing opportunities from pattern learning, adaptation, scalable distribution, self-organisation and self-healing from seemingly simple communications between swarms of simple, and not so simple, components. Deconstructing how these mechanisms operate, directly from studies of the human immune system enable us to uncover the governing principles of the biological immune system's engineering. With this knowledge these biological engineering principles can be applied to our own computational mechanisms.

Indeed in this study, earlier and guiding works from AIS research literature established detailed network traffic analysis systems (Harmer *et al.* , 2002), theoretical and architectural principles of immune homeostasis (Owens *et al.* , 2007) and conceptualised procedures to build and improve biologically inspired algorithms (Stepney *et al.* , 2005). From the developing works in AIS research and the fast progressing study of immunology we find interpolations, extrapolations and extractions of dynamic conceptual and novel viewpoints to many of our computational problems; including the challenging dynamic unknown representations in computer security to problem solving via infrastructural design, as addressed in this thesis.

Critical infrastructure, automation systems and industrial networks do pose unique security problems, with differences discerned from other more conventional computer and network security fields by many academics. Their problems carry the critical and easily understood element of risk to human life and the risk to the ways of human life. We largely focus on the rooted technical and infrastructural aspects of the problem that lead to the vulnerabilities in the, regularly ageing unpatched or unpatchable, control systems and their connected devices. Due to the often legacy state of devices in operation and the *above-all-else* requirement of real-time performance, these automation controllers are unable to run standard anti-malware solutions. These systems are big-responsibility, high-impact and as software attacks and fuzzing (hiding) mechanisms are already becoming more advanced, these critical infrastructure controllers are inevitably ever more attractive targets for advances on nation states, governments and corporations. Thus finding mechanisms to self-heal these systems in a non-intrusive yet reliable manner will be of great benefit.

1.2 SHAAM PhD Project

This collaborative PhD project between Aberystwyth University and EADS Innovation Works, later becoming Airbus Innovations, is entitled *Self-Healing Architecture Against Malware (SHAAM)* and was sponsored by the Electrical and Physical Sciences Research Council (EPSRC) and EADS Innovation Works. The project aims to build toward a system that can enable the necessary architectural robustness and autonomy to combat the novel and creative threat types, such as Stuxnet, that target programmable logic controllers, industrial networks and critical automation infrastructure.

The collaboration of ideas and objectives on the one hand influences the study toward applied research on real-world systems and on the other, toward the theoretical contribution to the Artificial Immune System (AIS) field and into the marginally more applied field of Cyber Security Research in Industrial Control Systems (ICS). This thesis reflects the study through problems and challenges in network security to specifically ICS security research, through the study of components of biological immunology and from bio to computational modelling in AIS to address those problems and through to scientific evaluation of decentralised and distributed systems on real-world computer networks.

1.3 Aims

This thesis aims to investigate self-healing architectures that will provide the detection and recovery mechanisms against the present-day threat of previously unseen malicious software attacks targeting industrial control systems and their networks.

From an academic perspective the aim is to investigate two holistic immune system metaphors. First the self-healing mechanisms that lead to its detection, recovery and resistance to common and new infections. Secondly its self-organisation that give way to collaborative, communicative, supportive, cognitive and homeostatic characteristics. The self-healing research direction carries, for us, a periphery interest to identify and correctly respond to unknown attacks while simultaneously learning a dynamic normal environment. With these items understood we aim to create and evaluate a stable framework that embodies those decentralised behaviours for application to cyber security problems and in real world networks.

The commercialisation perspective of this project carries an aim to achieve a proof of concept framework that encapsulates detection and recovery in a decentralised manner as suitable for application on industrial automation networks with safety-critical real-time systems, including low performance programmable logic controllers (PLC) or embedded PCs. Example settings include automated ore smelting systems, water treatment plants, manufacturing production lines and refinement process pipelines that use a Supervisory Control and Data Acquisition (SCADA) system with network attached PLCs, human machine interfaces (HMI), servers and workstations running the monitoring and control software. The infrastructural framework that is aimed for will give provision to use integrated security modules to detect malicious activity and to take response actions.

Software controlled automation systems are vulnerable to misuse, misconfiguration and malfunction of components and we know that the Stuxnet computer malware (Falliere *et al.* , 2011) caused and concealed malfunctioning hardware from an automation system's monitoring and control systems and thus its human operators. It is to identify and recover autonomously in these industrial environments with these kinds of previously unknown attacks of tomorrow that hold the ultimate aim for such an architecture.

1.4 Objectives

- To identify the key computer security challenges faced by industrial networks and their attached programmable logic controllers (PLC), see Chapter 2.
- To discover the key characteristics of the biological immune system that lead to its holistic self-healing, self-organisation and homeostatic behaviours, see Chapter 3.
- To assess the existing state of art research in decentralised self-healing system architectures for industrial network security, see Chapter 4.
- To design an immune system inspired proof-of-concept framework to address these issues, see Chapter 5 and Chapter 9.
- To design an evaluation framework to validate decentralised self-healing system architectures, see Chapter 6.

- To implement and evaluate the components of the designed framework for its immunisation rate performance properties, see Chapters 7 and 8.

1.5 Hypothesis and Research Question

The specific hypotheses addressed in this thesis are focused on testing an Artificial Immune System (AIS) architecture called CARDINAL-Vanilla that operates on computer networks. The architecture facilitates detection and response self-healing behaviours by distributing security information using a biologically-inspired heuristic as guided by many reactive components.

The key hypothesis that is answered in this thesis is:

> "In the context of a distributed and self-healing system, compared to engineered approaches a reactive multi-agent and holistic immune system inspired approach will have better distribution, and thus capability for self-healing, performance over a range of networked environments. Where performance is measured by a 'self-healing immunisation rate', consisting of the number of items transmitted, time to distribute an item and data sent to distribute those items, in order to assess the objectives of self-healing and application feasibility to industrial networks."

Which leads to the research questions:

> "To what extent will a decentralised self-healing security system inspired by an holistic view of the biological immune system, be able to distribute network transmitted security modules (i.e. detectors and repair tools) in a network of heterogeneous devices – such as an industrial control network? How will its self-healing and network transmission performance compare against an engineered system."

1.6 Key Contributions

The novel contributions of this thesis are in the following areas:

- A new viewpoint, theoretical architecture and evaluation methodology for the future of Distributed Self-Healing Security Systems (DSHSS) for protection of Industrial Networks and the Critical National Infrastructure that they will serve. Chapters 5 and 9 contain the developed theoretical architectures and Chapter 6 describes the evaluation methodology.
- The first distributed implementation of the Danger Theory-inspired AIS self-healing architecture CARDINAL-Vanilla, originally modelled by (Kim *et al.* , 2005). At this implementation level, we show its first mathematical instantiation in Chapter 5 and its first parameter tuning in two computer network environments in Chapter 8.
- In Chapter 7 we show, via a comparison under a comprehensive range of test scenarios, that the CARDINAL-Vanilla AIS architecture's immunisation rate is significantly poorer, yet without an important difference for a real-life applications, than our equivalent engineered self-healing architecture's performance.

- This thesis contains the first work of a decentralised Danger Theory-inspired AIS security architecture applied to the Industrial Control System (ICS) and SCADA cyber security problem domain and is among the first collaborative decentralised security systems to be applied to this problem domain.

1.7 Key Testing Areas in this Thesis

In this thesis the *CARDINAL-Vanilla* architecture is formally explained in Chapter 5 and is subsequently evaluated. The purpose of testing the new architecture is to extract specific and generalisable findings for future self-healing systems for distributed defence in SCADA and ICS networks. To achieve this the evaluations must prove the stability of the architecture, of the evaluation framework and show the feasibility of the architecture for wider application to industrial networks. The key measure of self-healing and feasibility in these evaluations is measured by our 'immunisation rate' metrics, defined in Chapter 6. The first testing area is a comparative analysis of prioritisation and distribution of security information. The second testing area is a sensitivity analysis of key parameters within the architecture.

The basis of the *CARDINAL-Vanilla* architecture was an abstract model without an instantiated mathematical or engineered definition. It embodied a complex hybridised system at the edge of research that combined agents, a distributed system, a distributed defence application, decentralised decision making, peer-to-peer connectivity and biologically-inspired feedback systems of decay, proliferation and suppression. In this thesis producing a stable version of the architecture and our evaluation methodology and framework has taken precedence and has led us to present these testing areas. Upon this base, our and other, future self-healing security systems can be extended and built.

1.7.1 Distribution of Security Modules and Benchmarking

Chapter 7 compares the AIS architecture algorithm against engineered algorithms on the key component that selects and distributes security information modules, containing detectors and responders. This comparative evaluation gives indication of the *CARDINAL-Vanilla* architecture's suitability to self-healing in the problem domain, over an engineered equivalent.

We show analysis of these tests under varying sizes of two network conditions. First is in a virtualised network running on a mid-range workstation and secondly on a single segment of an enterprise network. These experiments provide novel benchmark results of *CARDINAl-Vanilla* and in addition the first *fair* implementation of Kim et al.'s *CARDINAL* architecture with adaptations to an industrial network test scenario. This work shows the first use case of our self-healing security system evaluation framework described in Chapter 6 and further analysis of the performance discrepancies between the network types.

1.7.2 Architecture Parameter Tuning

Chapter 8 carries out a sensitivity analysis of the key parameters within the *CARDINAL-Vanilla* architecture. These evaluations analyses in detail the performance of discrete value ranges of the key parameters of the architecture, defined in Chapter 5.

These architecture evaluations are conducted using the validation framework in three parts. First range testing of the parameter value ranges, second an exhaustive step-wise search of the ranges in a virtual network and third a multi-start hill climb search under real enterprise network conditions. The outcome of this analysis is the discovery of a set of parameter configurations with good performance in our two network test types. These configurations can instruct the future application of *CARDINAL-Vanilla* into industrial networks.

1.8 Thesis Scope and Organisation

The organisation of chapters in this thesis carries the conversation from problem to solution, to evaluation of the platform component of that solution.

Chapter 2 explores broadly the problems that affect critical infrastructure and their industrial networks, with examples of specific cases. The chapter uncovers the existing security systems and standards in use to address these problems, while also reasoning why and where the limits of these approaches reside and concludes with a list of the main challenges.

Chapter 3 describes and discusses biological self-healing behaviours from immunology, cellular theory and beyond. In this chapter we uncover key characteristics of immune cells and their interactions that lead to the reactive and collaborative self-healing behaviours. The chapter concludes with mapping of solutions to the main challenges.

Chapter 4 positions the self-healing systems field and surveys the existing self-healing work under the Artificial Immune Systems (AIS) umbrella that focuses on related computer security problems. The chapter is concluded by selecting the existing *CARDINAL* decentralised abstract AIS architecture as a base platform upon which the solution is built.

Chapter 5 formally defines the mathematical model of *CARDINAL-Vanilla*, an extension of *CARDINAL*, as an immunity-inspired multi-agent platform for collaborative network-wide self-healing and security module distribution.

Chapter 6 explains how to evaluate decentralised self-healing security systems. This chapter accounts for key evaluation metrics, distributed user behaviour profiles, numerical objective maximisation, data sources and experiment procedures as required to evaluate distributed and collaborative network security systems, including our platform using key 'immunisation rate' metrics.

Chapter 7 addresses our hypothesis and research question by assessing the immunity-inspired distribution technique of *CARDINAL-Vanilla* against engineered and benchmark algorithms. These tests are conducted under a number of virtual and real-world network conditions. Together with the key results, in depth analysis of differences between the network conditions is provided. This work is the first of two use cases of the generalisable evaluation methodology presented in the previous chapter.

Chapter 8 contains a detailed system performance analysis of the *CARDINAL-Vanilla* model under differing parameter conditions. This work includes analysis of robust searches through the

parameter space under virtual and real-network conditions, and is concluded with a selection of several recommended and good performing sets of configurations.

Chapter 9 outlines the future work of this thesis as a result of carrying out this research. Presented is *CARDINAL-E*, a detailed theoretical architecture exhibiting key infrastructural components that address the set of main challenges specified in Chapter 2 and begin to address the greater challenges of adaptive malware raised in this project.

Finally, **Chapter 10** summarises the work done in the thesis, it presents our conclusions and emphasises points of future work.

Chapter 2

Security in Industrial Networks

This chapter presents a set of challenges in securing industrial networks and their necessary solutions that this thesis will address.

The topics covered include the purpose and description of industrial networks, the driving forces behind the necessity for their security (in section 2.1), the standards in place to achieve this (2.2), the technical state of the computer and information security field (2.3), the historical problems that have led to the current vulnerable state of industrial networks (2.4) and a summary of reported flaws in academic research (2.6). Another important topic is addressed by examples of recent targeted malware and the tools and opportunities that can easily lead to future malware in industrial networks (2.5). Prior to summarising the problems and solutions, the key case and positioning of decentralised and automated self-healing security systems is presented (2.7).

2.1 Introduction

Industrial networks are responsible for automation of process and production operations in facilities of national infrastructure, such as electricity, natural gas, water resources and road networks, as well as commercial industries including factory automation, ore smelting, chemical refinement, manufacturing, construction, aviation and in the transportation and distribution industries. These primary and production industries in the UK contributed to 14.4% of domestic gross value added (GVA) with a worth of £213.2 billion in 2012 (ONS, 2014) up from £205.2 billion in 2011 (ONS, 2013). Each of these industries depend heavily upon programmable logic controllers (PLC) providing the real-time operation via instruction-by-instruction control over their automated systems. The Supervisory Control and Data Acquisition (SCADA) systems are required to monitor and control those operations. These SCADA systems are connected to other administration networks and wider-area networks to feed data and information for decision making into other divisions of the organisation.

Protection of these critical infrastructures is undergoing increasingly intense debate in today's cyber threat-centric communities, particularly since the malware discoveries of Stuxnet, Flame and Duqu and the reports of ongoing advanced persistent threats (APTs) by various groups and nations (Binde *et al.* , 2011),(Mandia, 2013). While the days of "ever-changing self-sustaining malware" may not have yet reached us, the days of poorly secured architectures, unpatched zero-day vulnerabilities, APTs, targeted attacks and vulnerable national infrastructure certainly have.

2.2 Standards

Organisations are attempting to address the attack issues of critical infrastructure protection (CIP) at the industry sector, national and international levels. Each of their released documents offer detailed and comprehensive insights into best practice recommendations or in other cases, legally enforced regulations on system and information security policies. Some of the standards have been in draft since the 1990s. For example the North American Electric Reliability Corporation's (NERC) CIP-002 through CIP-011 standards documents have been released under five revisions with the latest approved in Nov 2013. The National Institute of Standards and Technology's (NIST) 800-82 rev 2 guide is by far the most accessible for newcomers to cyber security and also most recently released in May 2015 (Stouffer *et al.* , 2015). Figures 2.2 and 2.1 show an example Industrial Control System (ICS) network topology and an overview of the recommended "*defence in depth*" architecture strategy.

The documented recommendations provide practical implementation plans, guides and considerations for companies to employ current day technology and security techniques. A common strategy taken within the standards are to identify the critical systems as priorities, segment the network into functional "*enclaves*", apply a defence-in-depth strategy to secure each segment and then apply access control measures over those segments. If implemented correctly, the standards' approaches mitigate the risk and the impacts of successful attacks with a wide coverage over industries and organisations.

Our focus has not been on reviewing or picking apart the standards but in sharing an awareness of current security techniques used to prevent present day attacks on industrial networks. The standards spend years in writing and months under review for approval. They do not and cannot aim to be the only solution against tomorrow's attacks and hackers, but to provide a best-guess-framework for what may come. A list of the current standards designed toward securing ICS is in Table A.1.

2.3 Latest Vulnerabilities & Exploits

The cyber security specific standards provide strategies for system patching against newly discovered vulnerabilities. As with any released software these security flaws are in specific firmware, software, network protocol implementations, etc. and are found by security researchers through an intrusive and investigative process. Among the daily news feeds from the SCADA security research communities are frequent Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) reports on product-specific control system exploits, workarounds and patches. There is a rising trend in the proportion of found SCADA specific vulnerabilities since 2007 (fig 2.3a), with the biggest market share and majority of vulnerabilities affecting Siemens manufactured PLCs (fig 2.3b), as reported to the Open Source Vulnerability Database (OSVDB) (Kouns & Martin, 2015).

An infamous and exemplary security bulletin issued in August 2011 reported a flaw in Siemens SIMATIC S7 200, 300, 400 and 1200 series PLCs that permitted bypassing or disabling password protection on the devices (ICS-CERT, 2011). The S7 PLCs communicated with and remain using for interoperability purposes, the International Organization for Standardization Transport Service Access Point (ISO-TSAP) protocol which transmitted in plaintext passwords and payloads on port 102/TCP[1].

[1]In fact, ISO-TSAP is very similar to TCP. In other environments encrypted traffic that transmits with TCP at the

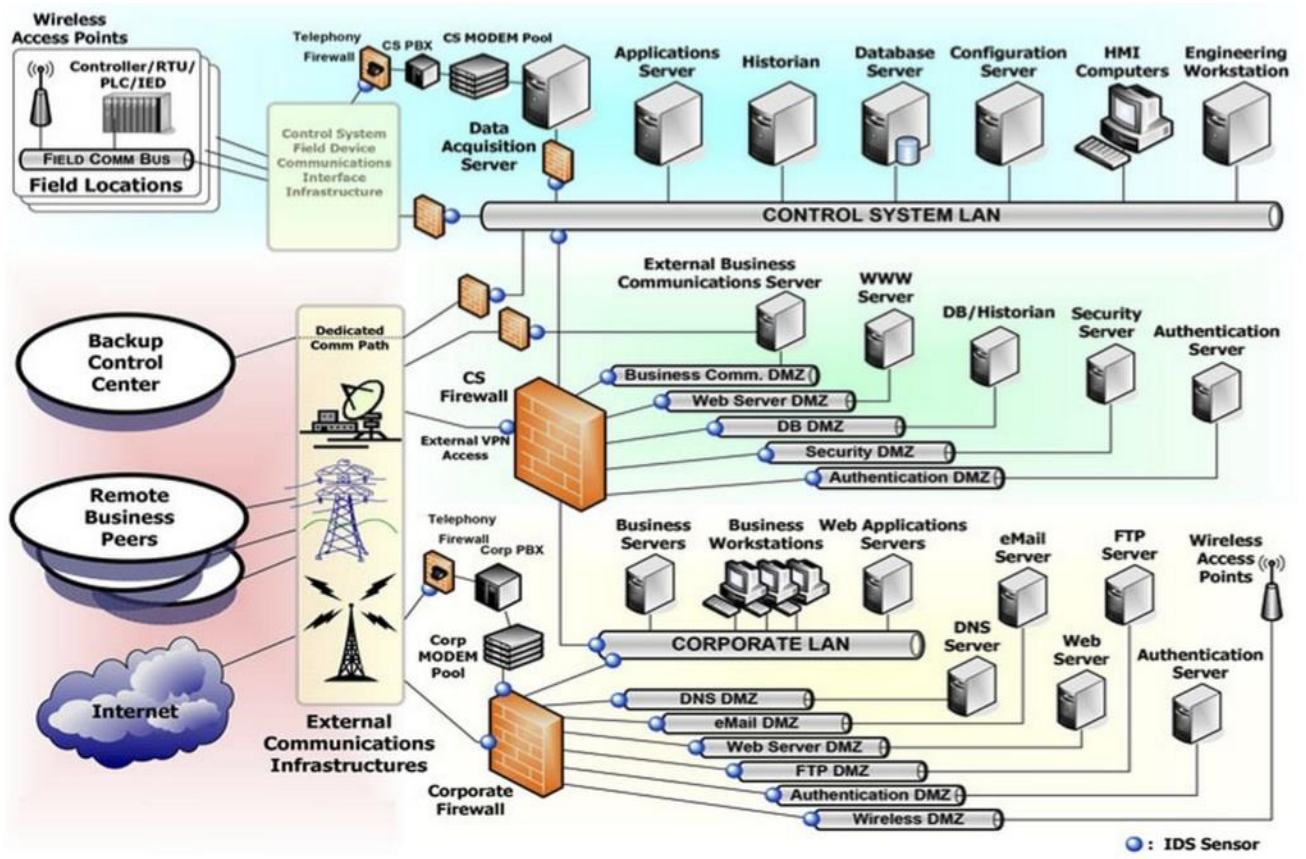


Figure 2.1 – Barrier-based *Defence-in-Depth* architecture, 2009 (p62), from NIST 800-82 Rev 2 Standard.

By replaying network packet content the 'takeover' is straightforward. In this case, Siemens released a downloadable firmware update for S7 1200 series PLCs. Infamy rises from zero updates made on the other devices and the ISO-TSAP protocol standard underwent no change in Siemens implementations. Instead the standardised mitigative workarounds were recommended, including monitoring and blocking of network traffic.

In March 2015, a hardcoded plaintext password vulnerability was found in Schneider Electric's integrated development environment (IDE) (ICS-CERT, 2015). In this case, the fix was to download and install a patch. In the Siemens S7 1200 case, the same applied. The firmware update for each PLC in a plant could be issued over the communication bus (often network), but not without first terminating the PLC driven automation operations. As late as May 2007 firmware upgrades ($\leq$v2.6.0) on S7-315 PLCs required installing the firmware update on a Siemens multi-media memory card (MMC) and executing commands from Siemens proprietary IDE. However this could not be done without first physically visiting and inserting the MMC card into each PLC (fig 2.4).

transport layer will encrypt its payload within the session to application layer of abstraction, i.e. by using secure sockets layer (SSL). That layer should be built into the S7 Comms protocol, but is not, or built into the receiving PLC modules.

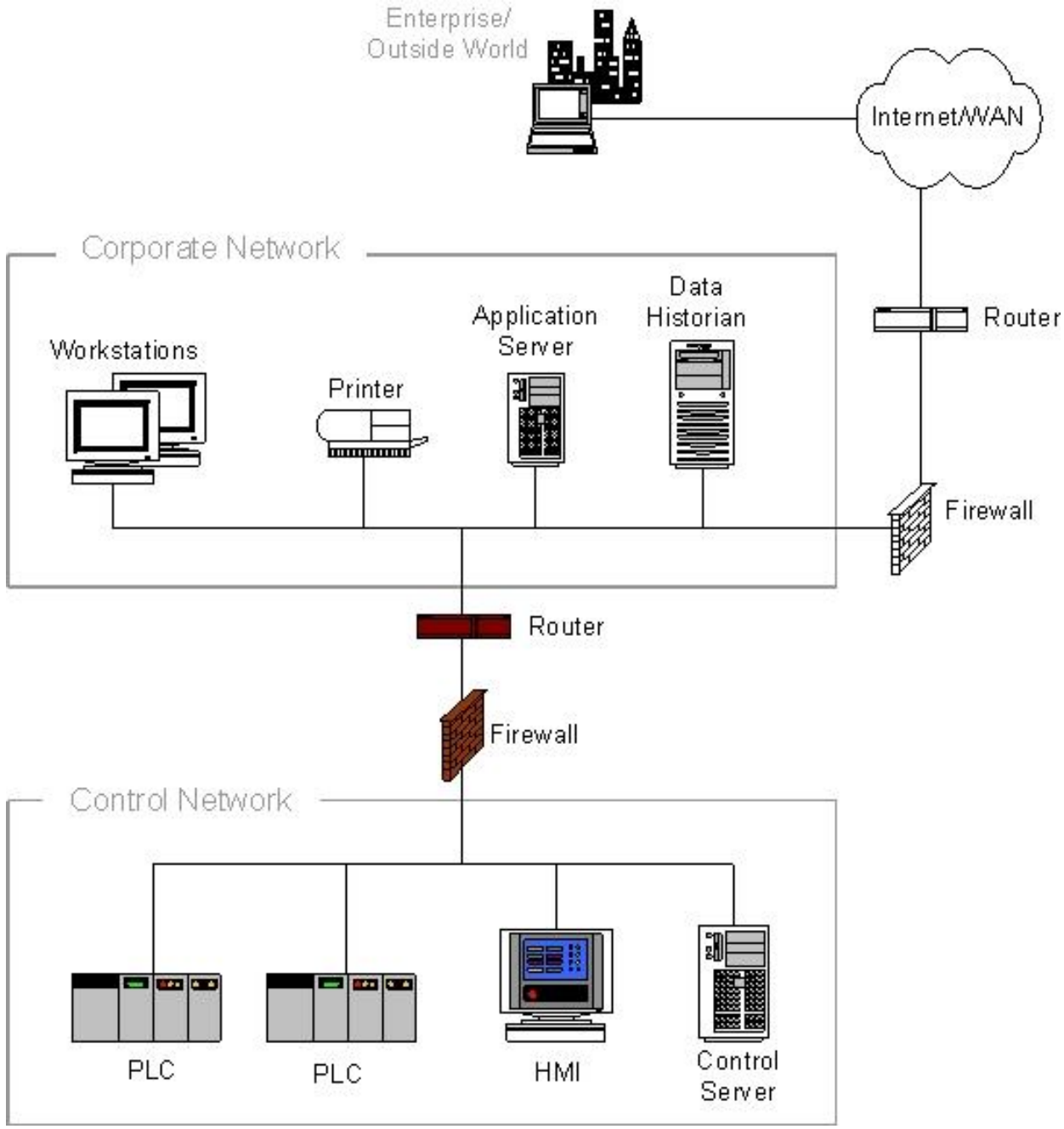
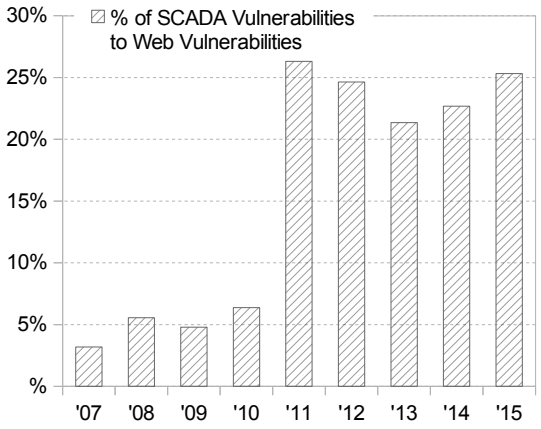
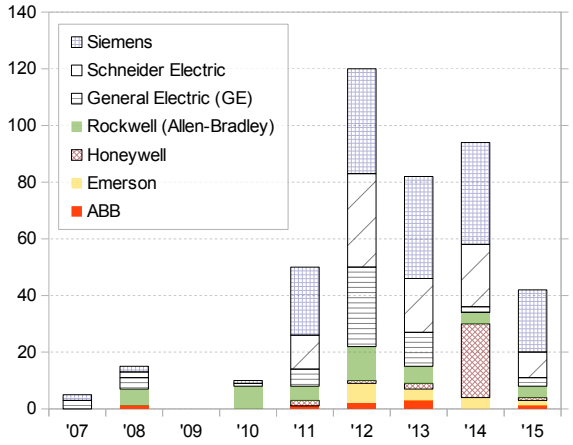


Figure 2.2 – Example of the corporate and control network topology division of industrial networks (p57), from NIST 800-82 Rev 2 Standard.



(a) % of vulnerabilities tagged "SCADA" vs. "web".



(b) Vulnerabilities per PLC manufacturer, stacks ordered by total reported.

Figure 2.3 – Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.2 and Table A.3.

Figure 2.4 – Automated chemical release and monitoring remote terminal unit (RTU) station. Powered by photovoltaic and long range communication via directional radio antenna. In a field on a hill. *Image by Dan Steele, 2012.*

2.4 Historical Issues in the Security of Industrial Networks

The threat of attack upon ICS and SCADA systems is a consequence of unfortunate evolving opportunity. But to understand the underlying causes of why industrial networks have reached this vulnerable state is to know a little of the history of PLCs and the automation industry.

PLCs are built for reliability and durability to operate for months and years without pause. Their life expectancy reaches into decades. Due to the initial investment of time and cost in development and training customers are often financially "*locked-in*" to their selected manufacturer's automation system. The generations of PLC product lines have undergone added functional modules and microprocessor/ memory hardware performance upgrades. This led to greater throughput at the expense of architectural changes and piecemeal growth of device revisions in an organisation's automation systems. The trade of maintenance complexity versus improved production volume is not taxing for financially driven management.

Systems builders and PLC manufacturers of earlier automation were concerned with physical security rather than information security. After all, early PLCs pre-dated modern day routable network infrastructure. Remote hacking of the systems was a zero priority as there was no physical or electronic route to the controllers, an "*air-gap*" as it were. Uploading new instruction binaries was via physically inserting a new memory card or via a direct and localised serial bus. Modern controllers still use these methods in addition to network received uploads. Systems were and have remained largely proprietary; however, defacto standards were introduced to enable interoperability between manufacturers, the ISO-TSAP protocol is an open example.

Business drove the demands for improved operational efficiency and real-time monitoring along with the widening use of interconnected networks, the Internet. The existing communication protocols were retrofitted to reliably deliver data over the new routable networks. The air-gap became logical via the use of firewalls. To support monitoring system interoperability, web server software were often integrated directly into the PLCs to enable convenient access to process state data and administrative utilities. TCP 80 port became an open target for *fingerprinting*. More widely used operating systems and their application software found their way from the corporate to industrial networks under the guise of cheaper and faster development of control and monitoring software and hardware. The standards, introduced above, were drafted, organisations tracked vulnerabilities, manufacturers took steps toward addressing issues in order to mitigate the security flaws as they arose.

2.5 Attacker Opportunities and Evidence

Over this period security researchers have grown in number, as have the training opportunities, investigative tools and revealed zero-day vulnerabilities, which are relatively easily found on industrial software and firmware. Tools such as Shodan (Matherly, 2009) map online devices using the nmap fingerprinting tool and allow users to search the internet for device addresses by tag name or vulnerability. Community driven and built open source penetration testing tools, i.e. Metasploit (Moore *et al.* , 2009), and open access exploit databases, i.e. Exploit-DB (Aharoni *et al.* , 2009), enable methods of attack by providing code and mappings from software version numbers to vulnerability snippets. The now ageing Zeus botnet framework (Binsalleeh *et al.* , 2010) and Metasploit's "Autopwn" feature, that

tests batches of exploits and on success installs an APT backdoor access, make for an attacker's delight. Building APTs are thus easy to access and manufacture, while zero-days get more commonly found on these less than appropriately secured and patched systems.

Despite best efforts, APTs can find their way onto the logical industrial networks. Whether delivered via open firewall ports and vulnerabilities or via social engineering and inserting a USB stick, the residing APT means total control over the workstation or server. From this point onward, if packing, fuzzing, obfuscation or other hiding mechanisms are employed the barrier-based defence-in-depth strategy is ineffectual.

On Siemens S7 automation networks the S7 communications proprietary application layer protocol and underlying ISO-on-TCP or ISO-TSAP transport layer protocols transmit traffic unencrypted. Open source tools such as S7Comms protocol disector for Wireshark (Wiens, 2013) or SNAP7 (Nardella, 2013) can be used to monitor and manufacture packets to the PLCs (see Figure A.3 for possible instructions). Alternatively, code from exploit databases can be used to issue commands to the PLC. Ultimately, if an APT is on the industrial network, then total control of the PLC is in the hands of the APT operator.

To monitor SCADA packets within the industrial network DigitalBond's rule-based QuickDraw plugin (Peterson, 2009) for the Snort network intrusion detection system (NIDS) (Roesch *et al.* , 1999) has recently been released and SecurityMatter's deep packet anomaly-based SilentDefence NIDS (Etalle *et al.* , 2014) has been commercially released in 2014. Both, however remain barrier-based blocking strategies.

There is evidence, that furthers this argumentation, from the Idaho National Labs (INL) 'Aurora' experiment where a controller-driven diesel generator was destroyed at the gasps of CNN reporters in 2007 (Meserve, 2007). The method used was to power on and off relays out of sequence, and has since been known as an 'Aurora-attack'. There are reports in 2010 apparently employed by a China-based group, describing its repeated use on large international US-based companies (McClure *et al.* , 2010). In the INL case, a vulnerability in a web browser was used to install an APT backdoor access in a development machine, the INL researchers issued commands over HTTPS to replace the controller instructions leading to catastrophe (Knapp, 2011, p37).

The Stuxnet malware was specifically targeted and marginally more autonomous; it was recognised by ICS-CERT in 2010 (ICS-CERT, 2010). It quietly damaged specific centrifuge products to hinder a controlled uranium enrichment process. It did so by replacing the spinning frequency to minimum and maximum values at random, while reporting normal values to the HMI and monitoring devices (Falliere *et al.* , 2011). It targeted certain geographical internet address ranges and several specific Siemens S7 series PLCs, using various zero-day mechanisms to propagated itself. Its hiding and attack systems have been the upper measure of cyber attacks since.

Stuxnet and Aurora are toward the more extreme of examples. However these impactful evidence-based attack reports are common in cyber security news articles and literature. They reinforce the issue of flaws and problems, despite best standardised efforts. They show that APTs can go unnoticed and that targeted attacks can take place even without receiving communications from a source.

2.6 Academic Summaries of Automation Security Flaws

Others have summarised these specific, highly specialised and technical vulnerabilities as fundamentally architectural, legacy systems without appropriate security capabilities, issues with SCADA network protocol standardisation, openness, i.e. use of open TCP/UDP protocols, authentication and access control issues, network misconfiguration and flaws in application software, (Pollet, 2002), (Igure *et al.* , 2006), (Ralston *et al.* , 2007), (Hong & Lee, 2008), (Ryu *et al.* , 2009), (Tsang, 2010), (Zhu & Sastry, 2010).

The focus is rarely drawn upon the need that we see for a collaborative lightweight security system that persists for the lifelong duration of the automation system through upgrades, additions and replacements of SCADA equipment and computer network infrastructure. Knapp raises the issue of "automated security software" (Knapp, 2011, p39-40), which is as close to the goals of our research as we have found. Knapp pragmatically forms the problem case for industrial networks to which host-based self-healing security systems are the solution.

2.7 The Case and Positioning for Automated Self-Healing Security Systems in Industrial Networks

To recognise novel sophisticated threats it is commonplace to combine contextual information, from multiple locations with multiple indicators, to make well reasoned decisions, as Figure 2.5 depicts. Enterprise network security practitioners use security information and event monitoring (SIEM) systems to collect varied contextual information from their workstations and other network devices. This approach is reflected in many of today's network security datasets such as the Westpoint ITOC 2009 (Sangster *et al.* , 2009) and DARPA KDDCup 98/99 (Hettich & Bay, 1999) and its derivative datasets and is recommended in published standards for ICS and SCADA networks (U.S. Department of Homeland Security, 2009), (Stouffer *et al.* , 2015).

Correlation and analysis engines can be used to model the collected data on a connected computing cluster in order to return modelled "*security intelligence*". The current state of practice is to visualise the received runtime log data against the learnt model data, as viewed from the centralised SIEM server. This visual information comes at an exorbitantly high cost, a slow pace and is information rather than

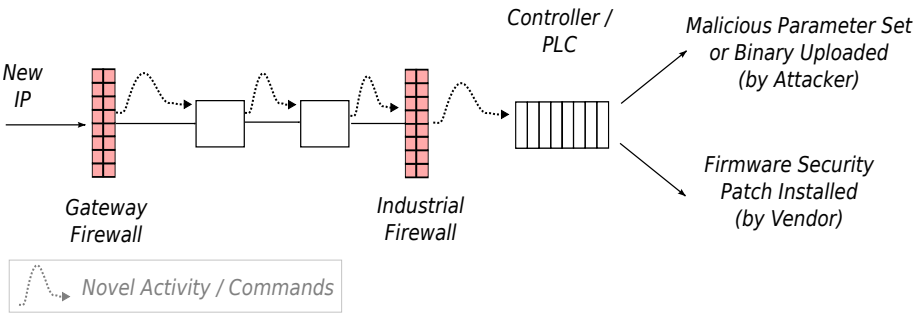


Figure 2.5 – Diagram illustrates a sophisticated attack with many innocuous steps. Showing single source of information is insufficient to distinguish a malicious attack from a benign update.

action [2]. A human operator must decide which incident response action to choose before manually initiating that response. This delays the process two-fold. First, is the transmission and receipt of the logs at the central server and second is the human factor. This delay is already inadequate to prevent today's attacks and certainly will be in the face of tomorrow's adaptive malware.

From our point of view, the decision engine's role is a search or specifically an unsupervised skewed two-class online machine learning problem. This search can be used to extract detection models, and separately extract recovery models, from runtime data. These lightweight models can then be distributed to harden the individual devices on the network. This action alone provides a collaborative and integrated full network defence system capable of acting in real-time. The ease and success likelihood of advanced persistent threat (APT) attacks and the life-threatening role that security systems play on industrial networks is sufficient to sway our judgement toward this collaborative and essentially host-based defence in depth methodology.

The specification requirements for such a system on ICS networks are complex due to many factors. These factors include the necessity for real-time detection and responses, zero-to-lightweight impact on the devices, safety-first responses, capable of running on legacy and cutting-edge equipment, capable of hardening systems with rare or intermittent connectivity and minimal error rates over the lifetime of the automation and security system. Aside from these complexities, our belief for why this system hasn't been created before are due to the relatively under explored academic research area of adaptive self-healing software systems for computer and information security (discussed in chapter 4), the data transmission expense of updating peer-to-peer knowledge systems and the state-of-practice machine learning approaches favoured in an industry where SIEM systems are needed.

2.8 Problems

This chapter has identified the following key problems that affect our current capability to keep industrial automation networks secure.

- There is a rising trend in reported SCADA vulnerabilities (stated in section 2.3).
- Applying patches or firmware updates can be infeasible leading to some vulnerabilities remaining unfixed or infeasibly-unfixable, i.e. the case specified affecting S7 2,3,400 (2.3).
- PLC customers financially locked-in. This caused parallel lines of vulnerabilities in the proprietary systems per solution, per PLC manufacturer as each retrofitted security patches and interoperability implementations (2.4).
- Industrial networks open to common vulnerabilities as they host common office application software and operating systems, either connected to the internet for updates or not connected and not updated (2.4).
- Management complexity increases as the systems change over time via upgrades, additions and replacements, i.e. in devices, software, protocols (2.4). Particularly applicable in businesses running long-standing PLC driven operations.

[2] McAfee's Enterprise Security Manager (ESM) SIEM product for SCADA networks costs $40,000 USD which offers "critical facts in minutes, not hours" (SCMag, 2015), whereas AlienVault's Unified Security Management SIEM product costs $17,700 for similar behaviours on enterprise networks (SCMag, 2014).

- Due to the above reasons, vulnerabilities are easy to find (2.5).
- APTs are easy to develop and create (2.5).
- Issuing new automation instructions is easy from within the control network (2.5).
- The network border barrier-based defence is ineffectual, if an APT is on the control network (2.5).
- Industrial networks operate within a dynamic landscape of threats and vulnerabilities as tools, technologies and security research expands (2.5).

The following is a summary of problems in the current state of practice for securing industrial networks.

- Standards provide a best-guess framework of principles for architectural security strategies for defence-in-depth, patching devices and implementations (stated in 2.2).
- State of IDS practice for SCADA and ICS networks use a network barrier-based approach (2.5), which becomes ineffectual in the case of meticulously constructed APTs (2.5).
- State of information enquiry practice in SIEMs use centralised logging and reporting; this approach cannot respond to incidences in real-time due delays from centralised logging and the human factor. The centralised design means that it cannot respond at sites if they have poor connectivity. (2.7).

2.9 Main Challenges and Directed Focus

These lists present the main issues identified based upon the problems specified in this chapter and our project motivations described in section 1.1. This following list informs our thesis architecture design, in Chapter 5:

- Self-healing (and self-hardening) behaviour.
- Host-based detection and responses.
- Distribution of security information.
- Independent decision making.
- Real-time performance, operate on modern and legacy devices.
- Minimal impact on device and network resources.

And, in addition to the above list, the following informs our future architecture design, in Chapter 9:

- Distribution of network-wide contextual security information.
- Real-time performance, operate on low level controllers.
- Independent decision making, under temporary or occasional network connectivity.

Chapter 3

Biological Self-Healing

This chapter carries the conversation through a precursory introduction into an holistic view of the Human Immune System (HIS) theories and specific behaviours of self-healing that are relevant to the previous discussion on engineering. Throughout this chapter correlations are drawn from the main engineering issues (2.9) to potential engineering solutions within the biological immune system. The chapter is concluded with a discussion and mapping of the immune system behavioural characteristics that address the engineering issues and lead into the following literature survey and our methodological approach in this thesis.

3.1 Introduction

The human organism is endowed with impressive natural defences against infection. Prior to advances in medical drug discovery, countless generations survived with only natural innate and adaptive protection against invading pathogens and antigens (bacteria, parasites and viruses). How bacteria, pathogen and viruses mutate to match our weaknesses and how the innate and adaptive systems detect and respond has undergone much study and remains an active topic of research.

While artificial immunology work has been under way for three decades, the Human Immune System (HIS) has undergone scientific investigations into inoculation and redefinition of the acquired immunity dating back to the 1700s (Silverstein, 2009; Jesty & Williams, 2011). In the recent sixty years the sustaining theories of immunity are the *Self Non-Self* (SNS) model (Burnet, 1959), the *Infectious Non-Self* (INS) model (Janeway Jr., 1989) and the *Danger Theory* model (Matzinger, 1994),(Matzinger, 2002). Each of these grand overarching theories have seen refinements and clarifications as the body of immunological knowledge has expanded. As a result immunology textbooks have been amended every few years with the new developments released along with new discoveries recognised daily in the scientific community.

The HIS descriptions in this chapter serve our purpose as a perspective snapshot of the human immune system. The content is directed based foremost upon Sompayrac's 2003 introductory text on immune systems (Sompayrac, 2003) and secondarily on Travers' 2008 'Janeway's Immunobiology' text (Travers *et al.* , 2008). To elucidate greater understanding reference has been made to Murphy's 2012 edition of 'Immunobiology' (Murphy *et al.* , 2012) and to Coico and Sunshine's 2009 'Immunology: A Short Course' (Coico & Sunshine, 2009), where cited.

In this section of work we will discuss two distinguishing components of the immune system, the *innate* system in 3.2 and the *adaptive* system in 3.3. This is followed by further discussion of aspects of biological homeostasis directly linked to immunology in 3.4. Each characteristic is extracted with relevance to our objectives and engineering problem.

A distinction between innate and adaptive immunity is commonly delineated and described as such; however, in reality both systems complement and are dependent of one another. From the perspective of an individual cell or a collection of cells there is no cognition that draws a differentiating line between the innate group's behaviours from the adaptive group.

3.2 The Innate Immune System

The innate immune system is the stock defence after the skin membrane layer and is highly effective against the common forms of antigens. Invertebrates, from the longest living animal – the Quahog Clam (McFall-Ngai, 2007), to the simplest and most thoroughly studied organisms – the *C. elegans* worm (Engelmann & Pujol, 2010) and to skates and sharks have survived relying only on innate immunity and without the lymphocytes and antibodies of the adaptive immune system (Beck & Habicht, 1996). This section describes some of the common behaviours of innate immunity in humans.

3.2.1 Tissue Immune Defences

First line tissue defence is provided by macrophage cells. Sompayrac estimates there are two billion monocytes at any one time in the bloodstream. Monocytes are an immature version of the macrophage, and in a ready state to mature and transfer into the tissue and muscle cells.

Macrophages are attracted to the fatty lipopolysaccharide (LPS) surface components common on bacteria. Their role is to consume and split bacterium and extracellular debris into smaller safe chunks. They use a process of *phagocytosis* in which the macrophage will swallow a bacterium in its phagosome vesicle (pocket). Its lysosome vesicle will then begin the process of *lysis* (enzymatic deconstruction or another form of degradation) upon the bacterium.

On encountering and consuming a bacterium, macrophages will release chemical signals (cytokines) to initiate a variety of effects. Firstly increasing the flow of blood to the region, then recruiting and communicating with more immune response cells. Inflammation occurs in non-trivial infections and in some cases sickness behaviours result (fever, sleepiness etc.). Interestingly, the latter has been found to potentiate the efficiency of the immune response (Aubert & Renault, 2008).

Key Points:

- Targeted detection
- Initiate communications to invoke response
- Generalisable pre-built response mechanism
- Revert to an offline mode (inflammation / sickness) to focus on recovery

3.2.2 Neutrophils Response

Of the traits of inflammation, neutrophil cells are probably the most apparent. Their main functions in the innate immune response are to ferociously kill bacteria whilst alive and form pus, the hallmark of inflammation, when they die. Neutrophils, of the granulocyte cellular line, lead very short lives and are the most common of white blood cells (leukocytes). Dead cells are quickly replaced by those circulating in the bloodstream; Sompayrac shows an estimate of 100 billion neutrophils produced each day in the bone marrow of healthy adults.

During the initial immune response in the tissue regions, specific cytokine messages (Interleukin-1 (IL-1) and Tumor-Necrosis Factor (TNF) proteins) secreted by macrophages will affect nearby blood vessels. These cytokine messages will invoke endothelial cells (the cells that line the vessels) to produce a protein on the surface of the inner wall of the blood vessel to which neutrophil cells are attracted and are permitted to pass through toward the point of infection. The transition only occurs when secondary signals are present in the region, i.e. the C5b complement proteins or LPS, as found on bacterium.

In summary, the neutrophils will slow at the area of infection and check for the second signal; only if it exists will they present the receptor required to attach to the area and begin delivering the response. In addition, the macrophages will regulate the release of message requests for reinforcement. This depends on the quantity of macrophage cells affected, such that greater infections reactively result in a greater request for reinforcement.

Upon arrival, the neutrophils will engulf the region of with the highest concentration of cytokines and bacterial infection. They will then react with either phagocytosis, degranulation or the neutrophil extracellular traps (NETs) responses (for the latter see (Brinkmann *et al.* , 2004)), choosing the response that best fits its situation to suppress or kill to the bacteria. The neutrophil will then die and form pus, which is reactively ejected from the body by other mechanisms.

Key Points:

- Secondary signal verification;
- Damage assessment and regulated recruitment directed by message release;
- Decentralised knowledge for localisation and mapping.

3.2.3 Complement System Responses

The complement system uses three immune response *pathways* to deal with antigen. Each pathway uses a two-step *recognise-and-respond* mechanism. Firstly a fine-grained receptor binds strongly to the antigen (bacteria, yeasts, viruses and parasites) and then secondly a component protein set will bind to the first receptor to then deliver a response.

The *lectin pathway* and the *alternative pathway* use knowledge of common molecules (amino, hydroxyl groups and mannose carbohydrates) found on the surfaces of most antigen. The immune network facilitates a set of *complement proteins* that circulate the bloodstream and are used similarly in both *pathways*. The *lectin pathway* uses another (lectin) protein to bind to carbohydrate surfaces on bacteria and to which the C3b protein will bind. In the *alternative pathway*, the C3b complement protein attaches directly to antigen with corresponding surfaces. After the C3b protein has attached, the response delivery system is the same. A stack of proteins called the membrane attack complex (MAC)

bind to C3b and will split open the bacterium's cell wall. The bacterium's intercellular proteins are then consumed and lysed by phagocytotic cells, such as macrophage cells. Thus neutralising the bacterium or antigen.

The *classical pathway* is the third type of complement system and relies on the attachment of antibodies to antigen. From the adaptive immune system, B-cells produce and release antibodies designed to benignly attach to the surfaces of specific antigen. Certain cells, such as macrophages, are attracted to the constant "heavy chain" regions of antibodies and either attach or begin phagocytosis. We will discuss this further in 3.3.

Key Points:

- One size fits most detection
- Abundance of cheap detectors
- Separated detection and response system

3.3 The Adaptive Immune System

The adaptive (or acquired) immune system consists of a collection of components that *adapt* solutions that can recognise new or unknown antigen and store these discoveries in *memory* for recurrent infections. The adaptive responses can take days to develop, but are more precisely designed and to detect antigen and viruses than innate components alone.

3.3.1 B-Cells and Adapting Antigenic Specificity

B lymphocyte cells are best known for generating and improving antibodies; the adaptive detectors for bacteria and other antigens. On their surface B-cells produce a uniquely formed receptor (BCR) with a light chain region that matches a specific antigen. Antibodies are a form of these receptors that detach and travel freely in extracellular plasma. B-cells are produced in the bone marrow and exist in several states, such as naïve and experienced with either immature or mature receptors regions. Their maturation and activation processes occur in several places including the germinal centres in the lymphatic organs. Activated B-cells differentiate into either antibody producing plasma cells or memory cells that remain in the body for many years.

The gene arrangement of their original BCRs, including their light chain (Lc) and heavy chains (Hc) regions occur by ordering and selecting at random from 4 types of gene segments. The subsequent gene arrangement is tested to ensure stop codon are correctly placed. The successfully tested Lc and Hc regions are then joined as an immunoglobin (Ig) IgM or IgD structure, and presented at the surface as BCR or secreted as an antibody.

Somatic hypermutation is the process of mutating or rearranging the gene segment template that produces the light chain regions of the B-cell's BCRs and antibodies. This fine tuning adjustment either improves the binding affinity to matching antigen or does not. However, those mutations that do improve binding also lead to greater proliferation and therefore those B-cells having undergone improved somatic hypermutation have a greater population than those with poorer affinity.

Isotypic class switching in B-cells leads to a modified heavy chain region in newly produced BCRs and antibodies. This occurs in mature activated B-cells when their CD40 and cytokine receptors are met by T-helper (Th) cells. This reactively leads to a BCR heavy chain region change from the initial IgM and IgD class types, to a variant of the five immunoglobin classes. Th2, Th1 and regulatory Th cells will release cytokines from the categories of IL-4, IL-5, IFNγ and TGFβ that lead to the change (Travers *et al.* , 2008). As with somatic hypermutation, the probabilistic behaviour in the body leads to greater populations of the B-cells that produce better targeted BCRs and antibodies.

The immune response is modularised by the heavy chain region of the antibody. The immunoglobin (Ig) categories have differing preferred locations, attract differing cells that cause differing effects; however, the common result is the ejection or destruction of the target to which the antibody had attached. The heavy chain regions of A,G and M attract effector cells with an appropriate (Fragment crystallizable (Fc)) receptor, leading to opsonising (covering), lysis via the *classical pathway* and degranulation via mast cells, basophils and eosinophils. Eosinophil cells are also activated by and respond via the antibody-dependent cell-mediated cytotoxicity (ADCC) process to immunoglobin E heavy chains, commonly associated to allergens and parasites. The G and M heavy chains activate natural killer (NK) cells via the ADCC process. Each of these mentioned cells are considered innate; this is another example of innate and adaptive immunity working together.

Key Points:

- Modularised detection, response mapping and responses
- Variable detectors, validated by probabilistic usage
- Variable class switching, validated by probabilistic usage
- Long term storage of detectors
- Modular adaptivity moves a functional unit from one component to another to enable new recognition, and as a result response, behaviours.

3.3.2 T-Cells: Dealing with Viruses, Regulation and Homeostasis

T lymphocyte cells are best known for their ability to recognise and kill the body's own cells that carry virus antigen without mistakenly killing uninfected cells. However, their homeostatic regulatory behaviours lead to truly intriguing supportive functions and reinforcement feedback loops.

T-cells are known to exist in two lineages as T-helpers (Th) and Cytotoxic Lymphocyte T-cells (CTL), their state is either naïve or activated. In their activated state they exist as Th0, Th1, Th2 and CTL cells, where they proliferate as plasma cells or, in fewer numbers, exist as memory cells. While activated they fulfil their functions. After their role has been completed, the redundant plasma cells undergo an activation induced death (ACID) to maintain an appropriate quantity for their need. The memory cell copies exists in some form for many years and can respond quickly to repeated matching infections.

Each T-cell has a receptor (TCR) on its surface. In a similar manner to the B-cell's BCR receptor, the TCR is believed to produce the wide variability shown by its receptor gene arrangement via selection from a small set of gene segments. T-cells and their TCR are known to undergo a survival test in the Thymus, restricted to TCRs that recognise non-self antigen presented in a major histocompatibility

complex (MHC) molecule. This *self-tolerance* procedure is the primary indicator leading to the *self / non-self* immunological theory. T-cells with TCRs that recognise MHC type I are classified as CTLs and have a surface CD8 receptor to improve binding to MHC-I. T-cells with TCRs that recognise MHC type II are known as T-helpers and have CD4 receptors the aid binding to MHC-II.

Naïve T-cells are known to become activated when an antigen presenting cell (APC) presents an MHC molecule and antigen that match the cell's TCR. They require a strong binding signal provided by a costimulatory signal. In T-helpers the binding of the CD4 receptor to MHC-II and the CD28 receptor to an APC's B7 receptor are known to aid the co-stimulation and the TCR binding. In CTL cells the CD8 receptor is known to aid binding to MHC-I. The T-cell activation process from naïve and activated is thought to occur after some state-dependent and order-dependent interactions between T-helpers, dendritic cells (DC) and the CTL cells. However, the exact process is "poorly understood" (Coico & Sunshine, 2009, p154) and similarly paraphrased by (Sompayrac, 2003, p65).

The T-helpers provide supportive functions for other innate and adaptive cell types. They do this through the release of corresponding cytokines. The Th1 cell will release IL2, IFN-γ and TNF cytokines, which lead to the proliferation of CTLs and natural killer (NK) cells; induce macrophages functions; induce B-cell isotypic class switching to IgG3 for opsonizing solutions against viruses and bacteria; activate primed macrophages and NK cells. The Th2 cell releases IL4, IL5 and IL10 cytokine proteins that lead to B-cell isotypic class switching toward IgA and IgE; these are solutions for mucosal, parasitic and allergenic antigen types.

T-helper cells provide recruitment feedback loops in tandem with other immune cells and are fed by the release of cytokines. Macrophages reactively release the IL12 cytokine upon meeting LPS bacterium. Th0, the multi-purpose T-helper, respond to IL12 by acting as Th1 cells. Th1 cells release IFN-γ which cause macrophages to activate and release more IL12 when in contact with other danger signals. Along with their supportive functions, T-helpers also provide self-regulatory functions. Th1 cells release IL2 in order to proliferate local Th1s, while also releasing IFN-γ to suppress Th2 cells. In opposition, Th2 cells release IL4 to proliferate Th2s and IL10 to suppress Th1 cells. These self-recruitment behaviours only occurs when their function is required. We know that T-helpers are required in a location when they serve a supportive or direct response function, therefore they reduce in quantity when they no longer receive the required signals. The physical voluminous geometry of a small infection causes this recruitment behaviour to be localised: As the distance from the source of infection increases, the concentration of danger and recruitment signals reduces; as the quantity of localised danger signals reduce, the concentration of recruitment signals reduce also. These spatially directed behaviours therefore lead to a state of homeostasis.

T-cells respond with the recruitment behaviour and finally upon a target cell with induced apoptosis. After activation via an APC the proliferating activated T-cell will increase its fluid capacity and thus probability of active effect in the body. When the activated TCR matches the MHC-antigen complex on the surface of a target cell, the T-cell (the CTL is specifically known to behave in this way) will release the Perforin protein to open the cell's membrane and will then deliver the Granzyme protein to induce an enzymatic apoptotic (programmed) cell death.

Key Points:

- Spatially dependent and multi-agent dependent regulatory behaviour

- Reactive module recruitment
- Probability-based rate of response and coverage
- Encapsulated object inspection and response
- Modularised detection and response
- Many adaptive components available in many (central) decentralised locations; not all locations are equal.

3.3.3 Antigen Presenting Cells

Antigen presenting cells (APCs) transport segments of antigen to B and T-cells which in turn can initiate an adaptive immune response. Macrophages, B-cells and dendritic cells (DC) are best known to present antigen (Sompayrac, 2003, p51); of which DCs are known to travel furthest, from the tissue to the lymph nodes.

DCs have several other important roles while in the tissue including recognising infections via "toll-like" receptors (TLR), phagocytosing and acting upon signals of danger. TLR4, 2, 3 and 9 receptors are known to be carried by DCs and recognise LPS, viruses, ribosome nucleotide amino acid RNA produced by viral infection and bacterial DNA (Sompayrac, 2003, p62). During these roles, DCs will ingest extracellular and cellular antigenic matter. The TNF cytokine will cause DCs to migrate and is released by macrophages when they experience bacterial antigen (Sompayrac, 2003, p48). The migration is routed along the one-way lymphatic vessels toward the nearby lymphoid organs or lymph nodes. Segments of the transported antigenic matter will be carried on the DC's surface in MHC molecules (MHC is defined in 3.3.2).

Lymph nodes are strategically distributed across the body and contain thousands of B and T-cells. The likelihood of activating a naïve B or T lymphocyte cell with a matching receptor for the antigenic payload is higher in these zones and increases over the few days of the DC's lifetime. Upon a match, the lymphocyte will proliferate and disperse via the lymphatic vessels. The result is a migration of ideally adapted B and T-cells for the infection. It can take several days for new antigen or less (several hours) for recurrent infections.

The APC behaviours contribute to further reinforcement feedback loops and give insight into the underlying conditions of the immune system that lend its homeostasis. Upon maturation, DCs release chemokine messages (similar to cytokines) causing recruitment of monocytes to develop into DCs and enter the area of infection. As the lifetime of the DC in the lymph node is limited, the new recruits reaching the lymph nodes provide up-to-date information of the infection; if DCs stop arriving at the lymph nodes, then no infection remains and a response is no longer required.

The immune response is proportional to the danger imposed by the infection as measured by the quantity of antigen being presented or cytokines and chemokines being released (Sompayrac, 2003, p49). However, some cells have differing functionality depending on the concentration of signals, i.e. a variable threshold, which may be the specific cause of the behavioural change. It seems likely that the physical navigation of B and T-cells activated by DCs to the source of infection is directed by a sustained concentration of cytokines or danger signals at the source of infection.

Key Points:

- Distributed decentralised resources of processing and for knowledge;
- Strategically located centralised units;
- Adaptive agent behaviour;
- Recruit reinforcements;
- Sustained proportional, simple and informative communications;
- Response proportional to severity assessment;
- Sustained communications direct and route the responses.
- Multi-functional agents detect, respond, communicate and transport matter.

3.4 Further Homeostatic Behaviours

This section explores other biological characteristics of an holistic view of immunology that lead to relevant homeostatic behaviours of the immune system.

3.4.1 Signalling

The immune system coordinates its actions in a conditional reactive decentralised manner and is guided by communication signals, Figure B.1 illustrates the signalling behaviour types.

The various signalling proteins, such as cytokines, chemokines and hormones, have signalling capabilities over short (paracrine) and long (endocrine) distances and internally (intracrine) and upon the sender (autocrine). The signalling between cells, as described above, has led to a complex yet deconstructable set of reactive behaviours that enable recruitment with targeted specificity, appropriate timing and concentration levels. Matzinger's danger theory (DT) (Matzinger, 1994; Matzinger, 2002) further proposed that control over immune system response is initiated via local recognition of *alarm signals* or danger-associated molecular patterns (DAMP). The subcomponents of cellular signalling, cellular receptors, inter-cellular interactions and the cellular membranes are proteins or chains of amino acid molecules. Thus for a given DAMP or protein if a match is discovered, an immune response will be initiated, irrespective of anatomical location or self/ non-self origin.

Key Points:

- Signals are broadcast to all nodes.
- Specific nodes listen for specific signals.
- Different signals travel different distances (signal specificity).
- Behaviour may change upon receipt of a signal. The decision for change is dependent on local state. The exact decision per cell is unclear.
- Context-based data recognition takes place, irrespective of the data's source of origin.

3.4.2 Two Signal Theory: Co-stimulation, Co-receptors and Binding Thresholds

Co-stimulation and co-receptor binding behaviours occur "reactively" between protein receptors across the immune system's components. The protein-protein binding is in most cases reversible and bound by non-covalent bonds. These, commonly, ionic bonds attract sets of oppositely charged molecules. The secondary receptors, among other benefits, give correctly charged molecules to strengthen the affinity in the localised region and thus owe to better receptor *specificity*.

Moreover, two signal theory is found in B-cell activation from naïve to mature, said to occur when the BCR binds to a corresponding antigen carried by an APC and while the B-cell's CD40 receptor binds to a Th cell's CD40L surface proteins (Sompayrac, 2003, p31). Another two signal theory example is virgin Th cell activation requiring stimulus from an activated APC. The Th cell's TCR and CD4 receptors bind to the APC's MHC class II molecule, together with a secondary stimulus binding the Th cell's CD28 protein to the APC's B7 protein (Sompayrac, 2003, p65). However, in molecular theory non-covalent binding affinity can be strengthened with any appropriately charged molecule. In B-cell activation, it is known that "clustering [.] a large number BCRs appears to partially substitute [.] co-stimulation by CD40L" (Sompayrac, 2003, p31). While this level of discussion fits our interpretative modelling purpose further intricacies of co-stimulation can be read in (Frauwirth *et al.* , 2002) showing many detailed extensions since the concept was first theorised in (Lafferty & Cunningham, 1975).

There are further cases of ligand receptor bonding discussed by (Cohen, 2000). Cohen states that *pleiotropia* is the condition where a single protein (gene) has multiple effects depending on whether its bonding is weak or strong. *Degeneracy* is the term used where multiple genes seem to perform the same function.

These examples show that binding, gene activation and thus immune cell decision making is non-binary. This is true at least at the level of cell-cell interactions and protein-protein interactions. The non-covalent bond decision threshold is variable depending on the local environment in which it is operating. Mapping non-covalent bonding directly to computational decision making requires a choice at which unit or scale to map ionic charges.

Key Points:

- Threshold levels that lead to a response can vary depending on local state.
- A threshold level can be exceeded in the event of a specific concentration (frequency) of proximate correlations (as opposed to direct correlations).
- A threshold's level can be exceeded when a direct correlation is found.
- In some cases, a threshold level will be exceeded only in the event of a logical conjunction of two or more correlations.

3.4.3 Evolving Self-Healing

The human biological system can change or evolve at various time scales. This includes at the birth of each new human generation, cell generation and at the level of somatic variation. An example of the latter are the B-lymphocyte cell receptors undergoing the hypermutation process (mutation and selection) in 3.3.1, leading to changed antigen recognition.

Two indicative examples of evolution at multiple time scales are the innate immune system's natural killer (NK) cells and, in a counterposition, viruses. According to (O'Leary *et al.* , 2006) and (Vivier *et al.* , 2011) NK cells have evolved to fulfil the role that is essential in the mediated fight against tumours and where the adaptive T-cells are unable to respond directly. Some pathogens and malignant tumour cells are known to downregulate or cease their cell's ability to express MHC class I molecules. CD8 CTL T-cells are thus unable to recognise and respond to the dangers of these cells. When tumour cells produce no inflammatory signals, T-cells further lack an activation pathway and instead consider the cell(s) as 'self'. NK cells have several mechanisms to recognise stressed cells and may respond via direct lysis of the cell or via cytokine release to invoke a cascaded adaptive immune response. In fact NK cells use NK-G2D, -p44, -p46, -p30 and DNAM receptors to recognise tumour cells (Terunuma *et al.* , 2008). In contrast, prostrate cancer tumour cells are known to shed NKG2D proteins that bind to NK cells and thus generate a false response, slowing the NK's effectiveness (O'Leary *et al.* , 2006). In a similar manner to T-cells, NK cells can also impose "immunological pressure" on the human immunodeficiency virus (HIV), but similarly to the tumour case the virus has changed to evade (Alter *et al.* , 2011). We therefore state that evolving self-healing characteristics exist within the system; be they generational in NK cells or reactive-variational as found by the HIV virus.

Darwin and Herbert's fitness measure of survival over generations is 'fittest', i.e. leaving most copies of itself that are suitable to exist in the current environment (Darwin, 1872; Spencer, 1896). The signalling mechanisms (see section 3.4.1) suggest that a cell and smaller entities, such as organelles, viruses or bacteriophages, are within a local (para-/ intra-/ auto-crine) environment and simultaneously in a remote (endocrine) environment; while also remaining embodied in a human. One can suppose, by these indications, that a measure of fitness applies at each of these scales. Section 3.2.2 exemplified regulated recruitment behaviours of neutrophil cells as a response to the signalling within an environment. This behaviour change is driven by some necessity. The combined effect of somatic variation, cellular signalling distances and regulated recruitment based on necessity, therefore indicate that evolving self-healing characteristics exist in different time scales, under a familiar fitness measure and driven by different environmental priorities.

Key Points:

- The system has evolves over generations and in response to pressure signals.
- Cells change to fulfil a required capability as indicated by signalling.
- The fitness measure uses the current environment state and *might be* based on prioritised need in local, regional and system-wide environment scales.

3.4.4 Cell Potency: Structural Adaptation and Organisation

Cell ageing, controlled (apoptotic) cell death and mitosis are natural biological processes that maintain a healthy internal state and contribute to the homeostatic behaviour in the body. The decomposed apoptotic cell 'bodies' are ejected from the body by phagocytes, i.e. macrophages. Cellular mitosis is a regular behaviour that replenishes dead cells over a duration, known to differ between cell lineages. Alongside these behaviours is an adaptive and controversially self-organising function, the cell potency of stem cells.

Cell potency is the potential for embryonic or somatic stem cells (in adults) to differentiate into different cell types. Somatic stems cells have been found in the bone marrow, brain, adipose (fat storage) tissue (Zuk *et al.* , 2002), cardiac cells (Beltrami *et al.* , 2003), in the third molars (Gronthos *et al.* , 2002) and are thought to reside in a 'stem cell niche' of each tissue (NIH, 2015). In each case of somatic stem cell potency, a progenitor cell will differentiate into one of multiple cell lineages, but unlike embryonic stem cells not all cell lineages. The bone marrow resident pluripotent hematopoietic stem cells, for example, differentiate into blood (white and red) cells and adaptive immune cells (Narasipura *et al.* , 2008). The differentiation may be induced by other somatic cells (already differentiated) that express only a few specific genes upon the stem cell's surface (Baker, 2007).

In the interest of *adaptive–above–adapted* systems we can find a correlation to computational role switching and even periodic role adaptation depending on the needs of the system. In the body, theoretically, under the correct guidance (expression of genes) one could envision dynamic structural changes and changes in cell or organ behaviours based on differing expressed concentrations.

Key Points:

- Cell function can adapt within a constrained range of roles.
- Cell role changes occur as a reaction to specific signals. The specific signal can also be due to some form of lack or abundance. The result is a homeostatic system.

3.5 Chapter Conclusions

In this chapter we have explored the key characteristics and principles of biological self-healing. Our holistic view upon the biological immunology has found complex networks of components ripe for application to intelligent decentralised decision making. The system maintains homeostasis and enables detection and recovery from infection while collaborating with other biological systems in its proximity. For these reasons the immune system provides interesting design templates for us to build self-healing and adaptive computational systems to address and deal with dynamic environments.

The key engineering characteristics of each biological topic have been mapped within the chapter and will not be repeated here. We have found that the immune system is composed of dependent, yet individual and absolutely reactive components that result in perceivably intelligent self-healing behaviours. The correlation to decentralised, distributed and self-organising applications, where the system has no single (central) point of failure, is a logical step. The biological signalling mechanisms enable self-managed communications and bottleneck avoidance, and can be developed into distributed signalling transmission behaviours. The adaptive, self-organising and *evolving self-healing* behaviours can be applied to Distributed Self-Healing Security Systems (DSHSS). First to generate adaptive detectors and responders via small variations based upon a localised viewpoint, such as in a host-based system experiencing its own view of a threat landscape. Secondly, in order to generate module-level changes based upon the system-wide threat landscape. Many of these engineering principles will inform the architectural design within this thesis, in accordance to the mappings stated below.

3.5.1 Mappings of Principles to Main Challenges

The end of chapter 2 specified the thesis's problem domain challenges (2.9). This chapter identified a number of applicable biological engineering solutions. Below is a mapping between the challenges (numbered) and their solutions (bullets). The 'not addressed' items are issues that remain unsolved by the principles extracted in this chapter.

Mappings:

1. Self-healing (and self-hardening) behaviour.
    - Neutrophils (3.2.2)
    - Complement System (3.2.3)
    - B-cells (3.3.1)
    - T-cells (3.3.2)
    - Evolving Self-Healing (3.4.3)
    - Cell Potency (3.4.4)
2. Host-based detection and responses.
    - Tissue Defences (3.2.1)
    - Neutrophils (3.2.2)
    - Antigen Presenting Cells (3.3.3)
    - Mature B-cells (3.3.1)

- Mature T-cells (3.3.2)

3. Distribution of security information.
    - Neutrophils (3.2.2)
    - Complement System (3.2.3)
    - Antigen Presenting Cells (3.3.3)
    - Signalling (3.4.1)
4. Independent decision making.
    - Neutrophils (3.2.2)
    - Complement System (3.2.3)
    - Signalling (3.4.1)
    - Two Signal Theory and Thresholds (3.4.2)
    - Cell Potency (3.4.4)
5. Real-time performance, operate on modern and legacy devices.
    - Not addressed.
6. Minimal impact on device and network resources.
    - Not addressed.
7. Network-wide contextual security information.
    - Antigen Presenting Cells (3.3.3)
    - Signalling (3.4.1)
8. Real-time performance, operate on low level controllers.
    - Not addressed.
9. Independent decision making, under temporary or occasional network connectivity.
    - Tissue Defences (3.2.1)
    - Antigen Presenting Cells (3.3.3)
    - Mature B-cells (3.3.1)
    - Mature T-cells (3.3.2)

Chapter 4

Artificial Self-Healing

This chapter opens by positioning self-healing systems and explores the state-of-the-art works in distributed self-healing Artificial Immune System (AIS) research with their application to industrial and computer network-based security. The chapter concludes by selecting a decentralised platform for use in the empirical trials of this thesis and as a foundation of a distributed self-healing security system designed to address the issues affecting industrial networks.

4.1 Introduction to Self-Healing Systems

Self-Healing Systems (SHSs) are software architectures that enable continuous and automatic monitoring, diagnosis and recovery of software faults by trying to eliminate and *harden* against the re-occurrence of a given fault. The resulting state of recovery is the key factor that contrasts self-healing from traditional fault-tolerant architectures. With that latter, recovery is to resume the state of execution and the former is to eliminate (or mitigate) of the fault's root cause (Keromytis, 2007). Ghosh et al. define self-healing as recovery from an abnormal (or 'unhealthy') state, return to the normative ('healthy') state, and function as it was prior to disruption (Ghosh *et al.* , 2007).

Several biologically-inspired approaches have been used to solve self-healing problems drawing on the common characteristic of homeostasis. Autonomic computing is one such approach, derived from the biological autonomic (involuntary) nervous system which controls many of the sub-conscious functions of the body. Recent research work on self-healing using autonomic computing can be found in (Huebscher & McCann, 2008). The biological immune system's capability to heal by identifying and reverting unsafe changes also follows the homeostasis theme. A survey on immune system inspired self-healing can be found in (Ghosh *et al.* , 2007), providing a more general exploration of self-healing than in section 4.2.

Software engineering approaches to self-healing also exist to solve these problems. These solutions more-readily consider fault tolerance, availability, resilience, survivability and dependability; and not always taking the decentralised networking view. However the same components of detect (monitor/ predict), diagnose and recover remain. The IBM manifesto for self-managed systems (Horn 2001 (Horn, 2001)) was carried forward by Kephart & Chess's 2003 seminal paper (Kephart & Chess, 2003) which cast the foundations for much of the work in self-healing and autonomic self-managed computing. In this realm detection is commonly via error monitoring and repair solutions are often interchanging of

or modifying input arguments into middle-ware. In the context of computational systems, self-healing systems use autonomous behaviour to address problems resulting from high complexity, a survey of this can be found in (Psaier & Dustdar, 2011).

Self-healing integrated operating systems and application language design have also been proposed and applied. Microkernel self-healing is grounded in delegation and management of child processes. On encountering an irrecoverable failure, the failed process must somehow be killed and restarted. A microkernel in any application (including operating systems) delegates tasks to sub-processes (threads). Those tasks that may endanger system stability under failure or that have a high probability of failing are chosen for delegation. These threads can then be monitored externally and safely restarted when required. An example is the Minix 3 operating system http://www.minix3.org. The Erlang actor model used by the "self-healing" AKKA distributed systems framework (from 2009 onward http://akka.io) is Ericsson's open source programming language developed for concurrency in clustered real-time systems. The AKKA actor model was designed to identify and localise faults using its hierarchical (tree) error-kernel. The error-kernel is fundamentally linked to the framework's microkernel and uses the refresh (restart thread), retry task (reissue function call) and report fault recovery options.

There are a number of other relevant and related fields to self-healing systems and Distributed Self-Healing Security Systems (DSHSS). Among them are self-adaptive systems (De Lemos *et al.* , 2013), swarm robotics (Brambilla *et al.* , 2013), self-aware systems (Lewis *et al.* , 2011), context aware systems (Baldauf *et al.* , 2007), socially attentive monitoring (Kaminka & Tambe, 2000) and trust management in distributed systems (Blaze *et al.* , 1999).

4.2 A Survey of Distributed Self-Healing Artificial Immune Systems for Network Security

This section explores existing works inspired by the human immune system that aim to solve security problems linked to the issues affecting industrial networks. After a survey of the state of the art, a gap was discovered. At the time of writing, there remains little other work on industrial network security problems exploiting inspiration from the AIS field. The works critiqued below were selected as they meet some or all of the following search criteria listed below:

- Distributed
- Decentralised control
- Self-healing
- Self-organising
- Network security focused
- Immune system inspired

The survey explores early topics in distributed network security and security systems covering early ground breaking works with immune inspiration. Then moves toward decentralised security system and decentralised holistic defence self-healing systems. These include self-hardening and self-healing security

systems employing multi-agent designs and decentralised cytokine signalling. Finally the influential topics and specific functional components are summarised into a set of subject areas.

4.2.1 Early Distributed Intrusion Detection Systems (DIDS)

The motivation outlined above can be generalised into a problem of autonomous and distributed network security. From this standpoint, Snapp et al. (Snapp *et al.* , 1991a; Snapp *et al.* , 1991b) in the early nineteen-nineties offered the first distributed intrusion detection system, named *DIDS*.

Snapp et al.'s expert system consisted of three agents running across the network. The agents included: a *host* agent service running on each machine to monitor its context data; a *LAN* agent process running on some machines to monitor network behaviour data; and a *director* agent process running on a single dedicated machine to analyse evidence reports from context and network agents and instantiating recovery mechanisms. In Snapp et al.'s solution the *director* agent also makes high level inferences respective of the reports' spacial-temporal information enabled by dynamic and static rule bases.

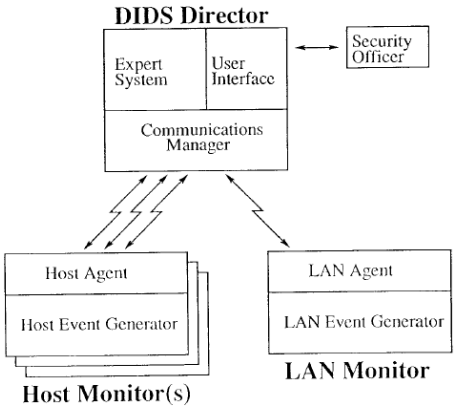


Figure 4.1 – Snapp et al.'s 1991 DIDS Architecture Diagram

Snapp et al.'s work in 1991(Snapp *et al.* , 1991a) set a standard in defining a blueprint for multi-agent and distributed network security systems of the future.

4.2.2 File Decoys and Negative Selection in DIDS

Marmelstein, Van Veldhuizen and Lamont were the first to move forward with a similar multi-agent blueprint in an immune system inspired context. In 1998 their publication (Marmelstein *et al.* , 1998) incorporated the single machine immune system inspired computer security work of Kephart et al. (1994) (Kephart, 1994) using a file decoy technique and Forrest et al.'s (1994) (Forrest *et al.* , 1994) negative selection detector algorithm both originally inspired by the *self / non-self* immunology theories (prior to Matzinger's danger theory (Matzinger, 1994))[1]. In doing so, they adjusted and combined the two approaches to reduce the inherent computational overhead in negative selection. First by reducing

[1]Broadly, T-cell lymphocytes mechanism theories for identifying viruses within infected biological cells describe a *training process* in the thalamus that discards new T-cells with receptors (TCRs) that have a high affinity for *self* amino acid strings and a virgin/mature T-cell *recognition process* that interacts and compare the amino acid strings (peptides) from other cells with their trained receptors. In the latter process a cell destruct (apoptosis) cytokine is released if a string sequence is recognised. Notably, this is also a cause of immunodeficiency diseases.

the area of file content from which non-self detectors were generated and secondly by only monitoring changes to the decoy files. The repair mechanism used changes to decoy files as distinct malware memory signatures which were then located and removed from other files.

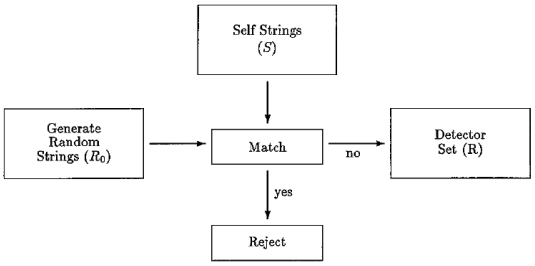


Figure 4.2 – Forrest et al.'s 1994 negative selection algorithm

4.2.3 File Decoys, Negative Selection and Grid Computing for Evaluation

Marmelstein et al.'s multi-agent architecture blueprint follows similarly to Snapp et al with three agents. A *local* agent monitors changes to decoy files on individual machines, parses changed decoy files with (negative selection-like) detectors and carries out file repair routines; a *network* agent classifies viruses using extracted signatures from infected decoy files and validated detectors; and a *global* agent extracts the signatures, generates detectors, evolves generic and virus-specific decoy files using a genetic algorithm and stores this knowledge. Explicit execution locations for these three agent processes is not described.

As with Snapp et al., scientific assessment of this architecture is impossible without their published results. However, Marmelstein et al. acknowledge that their and Kephart et al.'s approach to monitoring and detection is limited by its reliance upon malware that appends to files consistent with the decoy files and its dependency upon a decoy file becoming infected.

Marmelstein et al. also noted the problems encountered when drawing correlations between biological immune systems and computer architectures. Fundamentally they identified that biological immune systems are inherently parallel, whereas computer architectures are inherently sequential. That computer systems lack the "evolutionary adaptation mechanisms" found in immune systems and require careful application of evolutionary algorithms to fill this shortfall; suggesting "virus detection, virus purge [for example using an evolutionary search mechanism], and damage repair"(Marmelstein *et al.* , 1998, p3839). Due to possibility of file corruption caused during system repair they recommended a fail-safe process to recover file content. Finally alluded to was a distributed or grid computing approach to share the computational burden of evaluating detectors on the plethora of existing viruses.

4.2.4 Backup and Restore, Block Transmissions, Neuter Viruses

The architecture and lessons learned by Marmelstein et al. (Marmelstein *et al.* , 1998) at the US Air Force enabled the further enhancement of this distributed agent architecture and tested implementations (Lamont *et al.* , 1999; Harmer, 2000) culminating in the Harmer et al. 2002 (Harmer *et al.* , 2002) paper. Before discussing Harmer et al.'s scientific results, let's first cross the Pacific ocean and discuss the results of Okamoto and Ishida's 1999-2000 (Okamoto & Ishida, 1999; Okamoto & Ishida, 2000) similarly

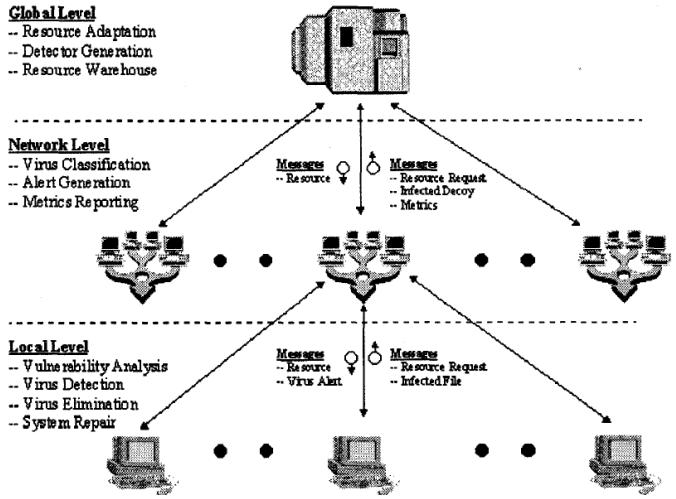


Figure 4.3 – Marmelstein et al.'s 1998 Architecture Diagram

themed distributed multi-agent architecture; then, with brevity, discuss a distributed application of Forrest et al.'s negative selection algorithm in New Mexico.

Okamoto et al.'s approach used four agents to perform virus detection, infected file neutralisation and file recovery. Agent communications permitted file backup and restore between machines and sending recommendations to other machines' agents to discontinue file transfers. We can deduce that all agent processes are executed on all machines.

The system included: a workhorse agent, the *antibody* agent to monitor change of infectable (executable) files under the —Windows— directory, analyse changes with *self* detectors (using a similar algorithm to Forrest et al.'s negative selection (Forrest *et al.* , 1994)), issue warnings to neighbouring machines causing them to block receipt of network file transfers from the originating machine and also neutralise infected files by replacing their HOF (head of file). Okamoto et al. extracted the *self* detector signatures from residing uninfected executable files. Additionally, a *killer* agent was used to remove neutralised files; a *copy* agent to backup uninfected files to other machines; and lastly a *control* agent to coordinate these agents.

Dissimilarly to the Snapp et al. and the Marmelstein et al. papers, experiments were conducted; although Okamoto et al.'s published results (Okamoto & Ishida, 2000) are missing conventional detail. The experiment compared detection, neutralisation and recovery mechanisms with equivalent mechanisms of "three commercial softwares (sic)". Anti-virus software names and versions were not given. The viruses tested created three file infection types: additions at EOF (end of file) and replacing HOF, HOF block overwrites, and dispersed fragmented overwrites (viruses were —Scream3—, —Joker2—, —Commander Bomber—, —AP-605— and —AIDS—).

A summary of their results reads as: commercial and Okamoto et al.'s detection mechanisms identified all viruses; commercial recovery mechanisms were mostly inadequate (0% success, 40% partial success), whereas Okamoto et al.'s were mostly successful for neutralisation (60%) and recovery (100%). Okamoto et al.'s results are remarkable and, from a distributed systems perspective, encourage further avenues of research to answer: would their system scale to today's commercial requirements of operating system and application size and versioning volatility? Beyond the distributed storage, can further advantage be taken of the distributed approach or the immune system analogies, such as sharing further

knowledge or delegating detector generation (as with Marmelstein et al.)? Can such a system adapt to newly added machines on the network? Can a file-based anti-virus framework like Okamoto et al.'s be applied to the network data transmission context?

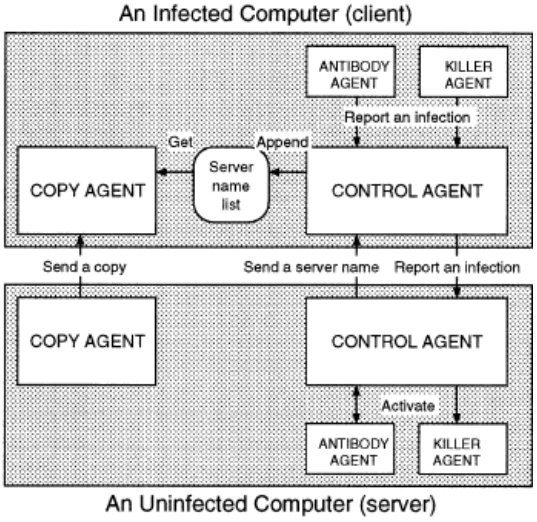


Figure 4.4 – Okamoto et al.'s 2000 Overview of Agent Control Mechanisms

4.2.5 Negative Selection and Decentralised Lisys

Forrest et al.'s self-non-self discrimination idea (Forrest *et al.* , 1994) was developed further in New Mexico during 1996-1998 (D'haeseleer, 1996; D'haeseleer *et al.* , 1996), (Dhaeseleer *et al.* , 1997), (Somayaji *et al.* , 1998) and (Hofmeyr & Forrest, 2000; Forrest & Hofmeyr, 2001) which included tests with intrusion detection systems (IDS) using the negative selection algorithm. An output of this line of work was the Lisys IDS software (Forrest, 2011; Hofmeyr & Forrest, 2000) embodying the negative selection algorithm refined by the research. Though, it wasn't until the Hofmeyr's (1999) PhD thesis (Hofmeyr, 1999) that the algorithm could be considered described in a distributed context and applied to network intrusion detection.

Hofmeyr explored two distributed architectures with an intrusion detection classification focus (Hofmeyr, 1999, p72-75). The first explored was the *centralised tolerisation*(CT) approach; that is, data are transferred from client (workstation) to server, detectors are generated centrally and then subsequently transmitted back to the workstations. Due to the communication overhead of CT, *decentralised tolerisation* (DT) was adopted. DT is the generation of detectors in situ (at the workstation) and avoids transmission of data for detector generation and the generated detectors between machines. This latter approach appears to retract from the advantages that complex biological systems, such as the immune system, offer.

4.2.6 Toward an Holistic Self/Non-Self-inspired Defence System

Now let's return our attention to Harmer et al.'s 2002 (Harmer *et al.* , 2002) paper. Harmer et al. developed upon the foundation that Marmelstein et al.'s distributed multi-agent security software architecture had laid. Harmer et al.'s implementation consisted of two groups of eight heterogeneous agents. The first group addressed file infections running on all machines and the second addressed

network intrusion behaviour optionally run at the network border (e.g. router). The eight agents performed the activities of their predecessors: an *antibody* agent to generate and store detector strings; a *detector* agent to detect file infections and network attacks; a *classifier* agent to identify malicious content; a *cleaner/killer* and *repairer* agent to recover; and *monitor*, *helper* and *controller* agents to report, communicate and oversee the other agents.

The detection systems of both agent groups are noteworthy. The file infection detection system used the negative selection algorithm (Forrest *et al.* , 1994) with a 16 bit detector string representation. The detectors were randomly generated, *censored* using static *self* data and matched during a scan to file contents; the system's intended *self* data is the full storage content, however, only a subset was used for experimentation. The network intrusion detection system monitored TCP, UDP and ICMP packets and used a 320 bit string representation (including protocol name, protocol data statuses, ports, source and destination IP addresses and validation bits). The network detectors were selected and matched against tuples of the MIT Lincoln Laboratories (1998 DARPA) network intrusion detection evaluation dataset (Hettich & Bay, 1999), containing genuine intrusion attacks. Both systems used the following matching algorithms: correlation co-efficient, hamming distance and landscape affinity matching. The landscape affinity matching algorithm used slope (change difference of two strings), euclidean difference and a physical (stacked string) difference (see (Harmer *et al.* , 2002, p257-258) for details).

Harmer et al.'s recovery mechanisms consisted of: quarantine file (move and change), delete file and "repair file" (not explained) for file infections and reset connection, route network packet to honeypot, block port, block IP address, and reduce priority of ("shun") IP address for the network intrusion detection system. Both systems logged actions and forwarded decisions and reports to a network administrator user.

The communication mechanism, used the AgentMOM framework (DeLoach, 2000) between agents running on separate machines, consisted of alert reports and detection threshold (sensitivity) updates (e.g. increase/ reduce). Let's consider the distributed characteristics of Harmer et al.'s system. Like Hofmeyr (Hofmeyr, 1999), detector generation was encapsulated at each machine. Unlike Okamato et al. (Okamoto & Ishida, 2000), Harmer et al. did not implement the distributed backup mechanism. However, Harmer et al. did exploit distributed and localised knowledge to increase the security awareness level of neighbours, in a finer-grained manner than Okamoto et al.'s approach.

Harmer et al.'s file infection experimentation used the TIMID virus (changes a random .com file

Agent	Goals	Services
Antibody	Generate, maintain, and store valid scan strings	Generate Graduate to memory Destroy scan string
Detector	Detect malicious code or network attacks at the input source	Scan input source Receive vaccination Update detection threshold Destroy antibody string Graduate antibody string to memory Get pointer to input source
Monitor	Coordinate the actions of a local neighborhood of agents	Receive information from a Controller Send, process, and receive alarm messages Receive warning messages Update detector detection thresholds
Helper	Communicate with the system user/administrator	Receive system information Receive costimualtion Receive action confirmation
Classifier	Implement the system response to an infection or attack	Identify malicious agent
Killer	Remove viral infections or network attacks	Delete malicious input
Repairer	Repair viral infections or network attacks	Repair malicious input
Controller	Coordinate global system operation Generate system operation metrics	Receive monitor status messages

Figure 4.5 – Harmer et al.'s 2002 Agent Mechanisms

in its own directory) and the EICAR test file (an .exe file that prints a string) for comparison with commercial security software. The network intrusion experimentation used labelled tuples of the MIT LL 1998 test dataset (Hettich & Bay, 1999).

Harmer et al. concede that their system lacked abilities necessary of an applied real-world distributed system fitting the computer security scenario (Harmer *et al.* , 2002, p277-8). Such a system would need to adapt to a dynamic self; that is, regenerate detector strings during runtime; authenticate and encrypt messages between agents; improve native time complexity inherent in the negative selection algorithm for file detector *self-selection* (pre-generation), *censoring* (post-generation) and file system scanning (16 bit string comparison) mechanisms. Use of additional system metrics (greater *context-awareness*); a mechanism to recover peers (network *self-healing*); and using adaptive detectors are suggested by Harmer et al.

4.2.7 Toward an Holistic Danger Theory-based Defence System

Swimmer's 2006 (Swimmer, 2006) paper moved toward a more complete distributed *autonomic defence network* (ADN) inspired by the immune system danger theory. Swimmer focused on functional industrial application of self-healing technologies to directly resolve the threat of malware, in-doing-so he described a number of relevant projects. First described were the algorithms of IBM's Anti-virus (1994), later used by Symantec, for novel virus signature analysis and extraction (partly patented by (Kephart & Sorkin, 1997), noted with brevity in (Swimmer, 2006, p1322) or detailed in (Kephart *et al.* , 1997; Kephart & Arnold, 1994)), detection, file recovery, automated signature verification, packaged release and finally replication mechanism to traverse across its hierarchical network. Second was the *Exorcist* project that used offline static binary analysis to model *self* (safe) system calls and integrated a kernel sensor to monitor process call stacks and process memory. Thirdly, proposed was tracking a buffer-overflow (BoF) attacked application crash to its attacker by correlating the application's crash dump report's buffered-string content to unencrypted time-relevant network packet content and then finally to the originating IP address(es). He indicated that a combined effort against the range of vulnerabilities was required by today's malware defence systems.

Swimmer's ADN model targeted network-based attacks via buffer overflow and via investigative tunnelling attacks making use of danger theory's analogies with *danger sensors* and *danger signals*. This used a malicious process detection system on all machines and a network perimeter-located (e.g. router) intrusion detection and firewall system. A functional description of the system was given, in place of the agent approach taken above.

The process activity detection system (PADS) consisted of process and service (daemon) monitoring using signature matching filters on inputs; additional sensors monitored selected prone (easily/commonly targeted) processes for "signs of distress" and used this monitoring data to extract attack signatures. Signature propagation is proposed via a network service discovery service update, after pre-registering all clients via broadcast. Proposed for signatures are a recency property (reduce evaluation priority over time) and signature (string) pruning via the TERIESIAS algorithm (Rigoutsos & Floratos, 1998). A removal mechanism for embedded malicious processes was not proposed.

The network activity detection system (NADS) is described as an "IDS or Honeypot" or as a "network choke point". Proposed is a semi-dormant process initialised by a message command (containing

a detector signature and a time-to-live period) sent from PADS running on the workstations. The NADS system will then parse network packets for these signatures for the given time interval. Blocks are applied to source or route IP addresses and combined with protocols and ports.

The distributed and communicative aspects of Swimmer's *ADN* model are (1) signature propagation, (2) raise signature priority and (3) PADS send signature message command to NADS. These, respectively, correlate roughly to (1) B-cell (antibody) proliferation and retention in the bone marrow (immunological memory), (2) cell with known antigen presenting in lymph node and (3) to danger signal cytokine/chemokine releases.

As with the other decentralised security approaches discussed above, thorough experimentation is not completed. For the purpose of our problem Swimmer's architecture is incomplete; however, his discussion incorporates a number of important considerations applicable to our problem. Firstly, within distributed systems on communications via network service discovery tools[2] and on trust (authentication) between nodes. Secondly that, without a complete collection of targeted defence systems (e.g. for viruses, buffer overflow attacks, botnets, rootkits and for novel attacks) vulnerabilities will remain exposed.

4.2.8 A Sting for Worms with Self-Hardening

Brumley, Newsome, Song and Pariente's (2005-7) *Sting self-healing* system (Brumley *et al.* , 2007) (Newsome *et al.* , 2006) (Newsome *et al.* , 2005) carried correlations to immune system antibodies (signatures) and antigen presenting cell traversals (vulnerability reports) to produce a four phase *self-healing* system against application exploit vulnerabilities. Unique among the works above, this system patched vulnerabilities of any application after a crash has been identified. With emphasis on exploits, the system falls short of a thorough defensive system but is an applicable contribution to such a system. Implementation of their model remains incomplete.

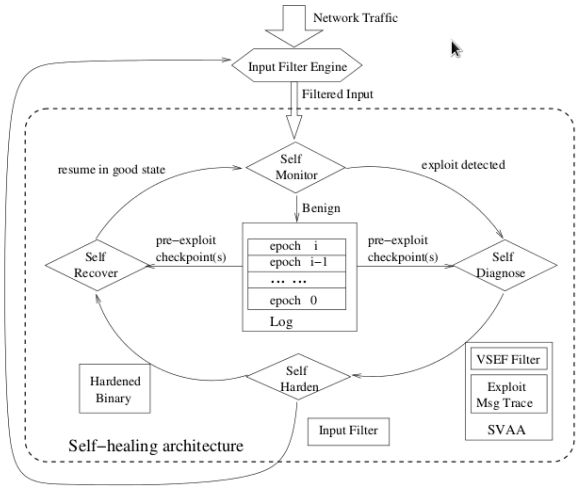


Figure 4.6 – Brumley et al.'s 2002 Architecture for self-healing process state

As with Swimmer, Brumley et al. (Brumley *et al.* , 2007) opted for a functional model description rather than the formerly common agent analogies. Their proposed model is described as: a monitoring

[2] As network service discovery tools Swimmer (Swimmer, 2006, p1332) noted ZeroConf (http://www.zeroconf.org) and Bonjour (http://developer.apple.com).

phase performed both logging and lightweight input filtering. A diagnosis phase performed non-realtime dynamic analysis of executables with a *replayed* log of system calls. System call log conditions are used to identify malicious calls and can invoke an automated *self-hardening* phase. This phase extracted a coarse vulnerability signature (parsed by the input filter with a tendency toward false-negatives) and an acute vulnerability detection binary of the logged series of calls (guaranteed not to produce false-positives) providing signature verification. The final recovery phase included process restart and process restart with a mechanism to *replay* wanted logged system calls.

Syscalltracker (Shalem *et al.* , 2003) provided continuous system call logging (system call log number, arguments and return values) and Flashback (Srinivasan *et al.* , 2004) provided periodic process checkpoint logging (stored process virtual memory, register values, open file handles, signal table, etc.), and the *replay* of logged system calls. As part of the system, address space layout randomisation (ASLR) is used to cause a stack (process) to crash, with a high probability, when the stack is attacked by a code injection, such as *return-to-libc* or *return-oriented-programming* attacks. Samples of inputs are also randomly selected for filtering in the diagnosis phase. The later diagnosis phase uses system calls and checkpoints to identify the input activity as malicious. The lightweight input filtering is not described further.

Diagnosis is implemented with a detection mechanism (TaintCheck (Newsome & Song, 2005) for buffer overflow, string-format and *double-free* attacks, also see related patent (Yeo *et al.* , 2010)) to analyse a process during a system call execution *replay*. The authors' implementation only parses binary files using Valgrind (though they recommend either PIN (Luk *et al.* , 2005; Luk *et al.* , 2011) or DynInst (Miller *et al.* , 2012)), however their detailed model is more interesting. The model's procedure reinitialises the process from a recent process checkpoint state (corresponding to the previous crash) and then conditionally *replays* the logged system calls. The system calls are tracked to the point where the input data caused the crash; this is then used to create the input filter and the system call data series (that lead to the crash) culminates into the guaranteed executable.

The model's hardening phase updates the signature database and distributes an *SVAA* (self-verifiable antibody alert), a packaged vulnerability test sample. The SVAA is proposed for evaluation (recreate the conditions and system calls that caused the process crash) running in a sandbox environment or virtual machine, the SVAA's signatures are discarded if the crash fails to reoccur.

Similarly to the diagnosis and SVAA evaluation, the authors' model recovery phase reinitialises the process from a checkpoint state and re-runs system call logs with the aim of making the process's internal and external state consistent. A crashed process may be safely resumed in a state prior to the attack. Replayed system calls are categorised into three types, where attacker *Type 2* calls may be given fabricated return values or ignored. Where correctness cannot be guaranteed the process must be restarted. In this case, their approach permits attackers' code, potentially retained in the external state (an external database), to be initialised during a secondary attack. With other checks in place this could be avoided.

The communication in this distributed system is minimised, only SVAAs are distributed. Each node independently makes decisions based upon the evaluation its own perception; as immune system cells operate. This decision gives redundancy and is thought may improve robustness. Otherwise put, each node (or cell) has no trust of other cells.

4.2.9 Danger Theory Modelled for DIDS

In 2005 Kim, Wilson, Aickelin and McLeod proposed the *CARDINAL self-healing* architecture to detect and respond to worm-based attacks, published in (Kim *et al.* , 2005). *CARDINAL's* architecture employed analogies of dendritic cells and T-cell tolerance. It was followed up by several extensions.

Fu, Yuan and Wang's added an embryonic intrusion detection framework (Fu *et al.* , 2007). Ou, Wang and Ou led an investigation for application to wired intrusion detection (Ou & Ou, 2010; Ou & Ou, 2011; Ou *et al.* , 2011b; Ou *et al.* , 2011a; Ou, 2012). Danziger and de Lima Neto (2010) extended the architecture to intrusion detection on wireless networks in recognising various WEP decryption attacks (*Cafe Latte* and *Korek's Chop-Chop*) (Danziger & de Lima Neto, 2010).

CARDINAL's approach is interesting due to the high-level correlation to collaborative diagnosis of danger. Upon Kim et al.'s architecture (Kim *et al.* , 2005), Fu et al. added a concrete weighted equation (Fu *et al.* , 2007) of the diagnosis metrics designated by Kim et al. Fu et al.'s prevailing diagnosis stated that, if a danger certainty threshold is not reached by the local client's evaluation of the weighted equation (attack severity, duration and detection certainty) the danger data will be forwarded to a central server for further evaluation. Fu et al.'s description implies dependency upon a single-point-of-failure, the central server. However, the probability of full-system failure can be reduced by simply increasing the quantity of servers; presenting the classic *performance-reliability* trade-off. A decentralised approach enables collaborative diagnosis without the single-point-of-failure problem.

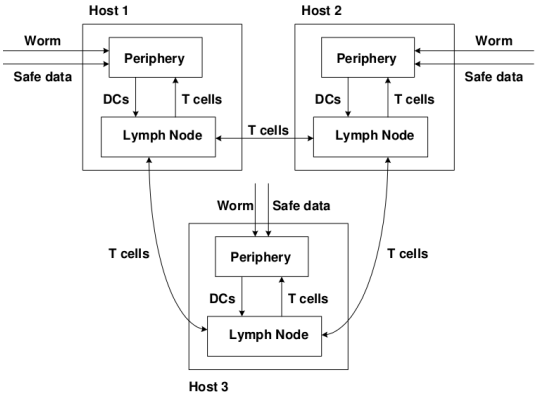


Figure 4.7 – Kim et al.'s 2005 Overview of CARDINAL (lymph-tissue) distributed architecture. Extended by Fu et al. 2007.

Fu et al.'s (2007) (Fu *et al.* , 2007) model framework focused on extending *CARDINAL's* danger theory-inspired architecture rather than targeting a specific security problem. Broadly, their agent-based approach monitored workstation context data and network packet data for intrusion detection.

The monitoring agent mechanisms used were described as *non-self* (anomaly) detection on system performance data (CPU, memory and disk utilisation), consistent with *CARDINAL*. The diagnosis agent mechanism evaluated the following danger metrics: (1) certainty of attack (2) severity of danger (number of agents reported same danger signal) and (3) duration of danger. Response types were determined by the normalised weighted equation with a three degree certainty threshold (uncertain; certain, resolve locally; certain, escalate to server). The recovery agent mechanism initialises if the attack is certain. The recovering client agent will disconnect network connections, then "create antibodies", reopen the network connections then "send these antibodies" to other agents. The detailed method of

recovery is missing from their description. Communications use the SCTP protocol over sockets, with *TELL* and *ASK* messaging in two channels (agent-agent and agent-server) to transfer "knowledge".

4.2.10 Cytokine Communications in Decentralised Defense Systems

In 2005 Guo and Wang (Guo & Wang, 2005) published their ideas toward an autonomous and decentralised security system framework using Mori's (1993) (Mori, 1993) broadcast communication mechanism (Concept and Data Field) architecture. Their short paper proposed applying Mori's communications architecture to common network components, such as router, firewalls, switches and also servers and workstations. This application indeed correlates to intracellular cytokine messages released (broadcast) by immune cells and where only specific cells respond (e.g. a multi-recipient channel). Mori's (1993) (Mori, 1993) autonomous fault tolerance architecture is based on control and coordination enabled by precise communication; the *data field* component indicates that all subsystems (nodes) broadcast their data, however only nodes registered to the *data field* listen and respond to these broadcasts.

4.2.11 Intrusion Prevention and Multi-Agent Self-Healing

Elsadig, Abdulla and Samir (2009-2010) (Elsadig & Abdullah, 2010; Elsadig *et al.* , 2010a; Elsadig *et al.* , 2010b; Elsadig & Abdullah, 2009) propose a modelling approach toward an artificial immune system host-based intrusion prevention and *self-healing* system. In (Elsadig & Abdullah, 2010; Elsadig *et al.* , 2010b; Elsadig & Abdullah, 2009) the abstract model framework of four agents (*Sense*, *Analysis*, *Adaptation* and *Self-Healing*) is based upon *dendritic cell—lymph node* interactions. An analysis of dynamic agent state transitions (using Petri-nets and formal notation) is given and discussion of model refinement using the Stepney et al (Stepney *et al.* , 2005) framework. A multi-agent implementation is described in (Elsadig *et al.* , 2010a) performing input monitoring and anomaly detection (*Sense*), unsupervised classification for diagnosis (*Analysis*) and classification model updating (*Adaptation*) described using *Cluster-K-Nearest-Neighbour*, *K-Means* and *Gaussian Mixture Model (GMM)* and recovery (*Self-Healing*), however no validation was published. Agent interactions follow the FIPA standard (Elsadig *et al.* , 2010a; Elsadig *et al.* , 2010b). The core signals are: send *anomaly detected* signal with behaviour data, send *misuse behaviour data* and send *modelled misuse behaviour data*. The agent interactions are described between agents on a single node.

For our purposes, several limitations befall Elsadig et al.'s proposed model. Missing, vitally, are concrete declarations of R'', the set of abnormal behaviour for each *category* (monitored data context), and $f'(sys)$, the function reversing the system state from anomalous to normal. Their model is not distributed across multiple nodes, the implementation remains incomplete and validation against a specific malware target has not been published. However, the comprehensive agent component of Elsadig et al.'s self-healing framework can be extended to yield a decentralised system.

4.3 Chapter Conclusions

This chapter has surveyed the artificial immune system (AIS) field's academic works on self-healing and distributed systems with a focus on security issues related to our criteria. Below are the key findings of this review on self-healing components including detection, diagnosis, prevention and recovery and on relevant findings on distributed systems including communication and security mechanisms.

Detection and Diagnosis

The detection processes from artificial immune systems (AIS) discussed include a simple file decoy approach to detection, diagnosis and identifying virus cures have been introduced by Kephart et al. and modified by Marmelstein et al. A negative selection algorithm for self/ nonself (anomaly) detection by Forrest et al. has been used by Marmelstein et al., Harmer et al., Hofmeyr et al. and Okamoto et al. Detection designed with conventional engineering and software vulnerability considerations include application crash dump analysis systems introduced by Brumley et al. and Swimmer. Signature-based diagnosis are used by Marmelstein et al., Harmer et al., Swimmer, Brumley et al. Classification/clustering mechanisms for diagnosis were used, rarely among the works reviewed, by Harmer et al. and Elsadig et al.

Self-Hardening: Sharing and Trust

A static *self-hardening* (future prevention) process was introduced by Brumley et al. which updated diagnosis input filtering mechanisms. Brumley et al used an *SVAA* packaged approach to signature distribution and verification from untrusted sources. Whereas, Kim et al. and Fu et al. with CARDINAL and Marmelstein et al. and Harmer et al. gave an abstract description of signature distributions with trusted sources. Guo et al. described application of Mori's broadcast and selective listening approach to signature distribution. Anomaly detection threshold sensitivity signal alerts were distributed by Harmer et al. and Elsadig et al. to increase neighbours' sensitivity to/ awareness of danger. Hofmeyr's PhD thesis took an alternative approach first exploring *centralised-tolerance*, a centralised store of signatures, however finally selected an entirely *decentralised-tolerance* approach with an independent and individualised set of signatures that were not transmitted to neighbours. Swimmer chose an attacker track-back approach, matching host infection signatures to historical network packet data.

Recovery Mechanisms

Recovery mechanisms may include signature identification, signature extraction, signature verification and application of a signature-based removal mechanism. Mechanisms to respond to process attacks, file infections and detected network attacks from generic malware types have been described. Brumley et al. used restarting the process and *replaying* systems calls to restore the process's state, using crash dump state and system call log to identify the attack. Process monitoring to extract signatures and network packet monitoring were integrated to block network IP address, protocol and port combinations by Swimmer. Swimmer also describes an IBM patented approach to signature extraction of a virus infected file; stating that, in most cases, the virus infection is reversible where a genuine-patch is not. Okamoto and Ishida used network stored file backup and restore. Others, such as Harmer et al., used

traditional anti-virus mechanisms, network IP blocking/ priority reduction and forwarding packets to a honeypot.

Collaboration and Communications

A collaborative network perimeter-based and host-based prevention system is introduced by Snapp et al., Marmelstein et al., Harmer et al. and Swimmer. Swimmer recommended Bonjour and Zeroconf network service discovery tools for update distribution; which can also be extended to initialise the self-healing software on newly connected machines.

4.3.1 Discussion

This state of art review has explored works on network security and network defence from malicious software which take inspiration from immunology and its theories. While no works on industrial network security have fallen under the immune system inspired search criteria, the survey has uncovered a range of emergent features from the artificial and biological systems, including self-healing, self-hardening and homeostasis functions that can be exploited in future designs of architectural solutions.

4.4 The Road Ahead

The survey has shown that a gap exists for works that apply immune system inspired decentralised self-healing to industrial networks and the issues we have raised. To address these problems, a range of new infrastructural support components are in need of development, as we have summarised in this chapter's conclusion section.

The first stage to build this architecture was to select, build, evaluate and optimise the decentralised platform of the system that will facilitate the self-healing and security module distribution tasks. We have chosen to use the *CARDINAL* abstract model architecture by (Kim *et al.* , 2005) as the foundational platform for a number of reasons. We introduced and critiqued the model and its extensions in 4.2.9.

CARDINAL is an Artificial Immune System (AIS) designed as a networked software architecture with peer-to-peer connectivity for network security applications, in particular it has the foundations of a Host-based Intrusion Detection System (HIDS). It is modelled on the recent and developing immunological theories of Danger Theory (DT), T-cell decision making and cytokine signalling via concentration, reminiscent of frequentist probability. Its modelling of the immune system is similar to our own viewpoint in Chapter 3. Its decision making technique for handling unknown inputs is similar to the well known and tested Dendritic Cell Algorithm (DCA) (Greensmith *et al.* , 2006), using frequency as an indicator for belief. Unlike the DCA, it is distributed and decentralised and has no published scientific evaluation at a distributed implementation-level. Additionally it models transmissions of detector and recovery modules, employs a conceptual classifier with individually- and collectively-built models. Thus we believed it to be ripe for testing and adaptation to our application as a self-healing security system for industrial networks.

The next chapter will describe our mathematical model of *CARDINAL-Vanilla* from an implemented perspective of *CARDINAL*. The next chapter will present how to evaluate decentralised self-healing security systems, including *CARDINAL-Vanilla*, and is followed by experimentations on the architecture.

Chapter 5

The CARDINAL-Vanilla Architecture

5.1 Introduction

The CARDINAL-Vanilla architecture is our mathematical instantiation, implementation and clarification of CARDINAL, Kim et al.'s artificial immune system (AIS) abstract model (Kim *et al.* , 2005).

In summary, the architecture is designed as a distributed self-hardening system. Inputs are classified and mapped to a response – the *signature*. That signature is contained in a *signature module* (an *agent*) and is transmitted to all nodes running the networked software application to then provide an immediate response to that input. Each node runs an identical software client of the networked application; the holistic view of which is the *architecture*.

The CARDINAL-Vanilla architecture is formed from the same biological analogies as Kim et al.'s model; however, many of the model details, including parameters, their values and ranges, were omitted from publication. Both Kim et al.'s and our models are inspired by biological metaphors and are defined as multi-agent based and decentralised computational decision systems. Kim et al.'s model was intended to prohibit email spamming malicious software in a distributed defence form. Our final intention is an holistic distributed self-healing system using behavioural monitoring to generate detection classifier models and recovery tools using integrated and transmittable *signature modules*. Our critique of their model is in (4.2.9).

The most valuable contribution of this chapter is the explicit definition of a novel probability-based heuristic for transmission of high priority information in decentralised peer-to-peer network applications, stated in (5.10) and (5.11).

Other contributions of this work are specifying the mathematical model, specifying a number of key differences and clarifications between the two models (5.2) and the chapters of experimentation and evaluation that follow. The experimentation chapters investigate the effects of the key architecture parameter ranges (5.12) and will focus on the measurements pertaining to network-wide immunisation, which is particularly addressed by the signature module distribution decisions (5.10) and their priority heuristic (5.11).

In this chapter we will define the processes (5.4) within CARDINAL-Vanilla and its flow control (5.5), the network communications (5.6), classification of inputs, their responses and signature module creation (5.7), signature module distribution decisions (5.10) and their priority heuristic (5.11), the time scale in which the agents interact and operate (5.9), the confirmation process of attack inputs

(5.8) and the key parameters of the architecture (5.12). We will discuss each of the key behaviours with respect to our modelled computational decisions and agent interactions, see (Kim *et al.* , 2005) for the original immune system modelling description.

5.2 Clarifications and Differences

The CARDINAL and CARDINAL-Vanilla architecture models are intentionally similar. The intended cause for difference is due to the change to target domain, from email spamming on enterprise networks to self-healing on industrial networks. Implementing the model as a networked software application identified underspecified algorithm, process and mathematical modelling details. This chapter specifies those changes.

- Table 5.3 on page 85 details CARDINAL's textual definitions that required an engineer's interpretation. The adjacent column contains the CARDINAL-Vanilla model definitions, as implemented, and as guided by the immune system inspired principles specified in chapter 3.

- Table 5.2 on page 84 contains the newly specified mathematical model parameters and variables. These data items, default values and ranges did not exist in earlier literature. These are new for both CARDINAL and CARDINAL-Vanilla models. This definitive list of parameters and variables can be referred to while reading through the mathematical instantiation described throughout this chapter.

- Figure 5.3 on page 65 is the process diagram of CARDINAL-Vanilla in an equivalent representation to Fig. 2 in (Kim *et al.* , 2005). The two diagrams illustrate the process similarities and differences in the structure of interactions, in process and data representations, including that of agents as processes, and in minor process structural changes. The numbered processes and their differences from Kim's model are stated in section 5.4 on page 64.

- Figures 5.4 and 5.5 on page 68 illustrates the flow control of the Tissue and Lymph Node processes within CARDINAL-Vanilla. These are presented in an equivalent representation to Fig. 3 in (Kim *et al.* , 2005). The diagrams illustrate the flow control similarities and the differences in specificity, complexity, vital simultaneous execution and order. The flow decisions are described in section 5.5 on page 67.

5.3 Architecture Overview

This section will introduce the CARDINAL-Vanilla architecture from the perspectives of engineering, its immunological inspiration and its multi-agent modelling.

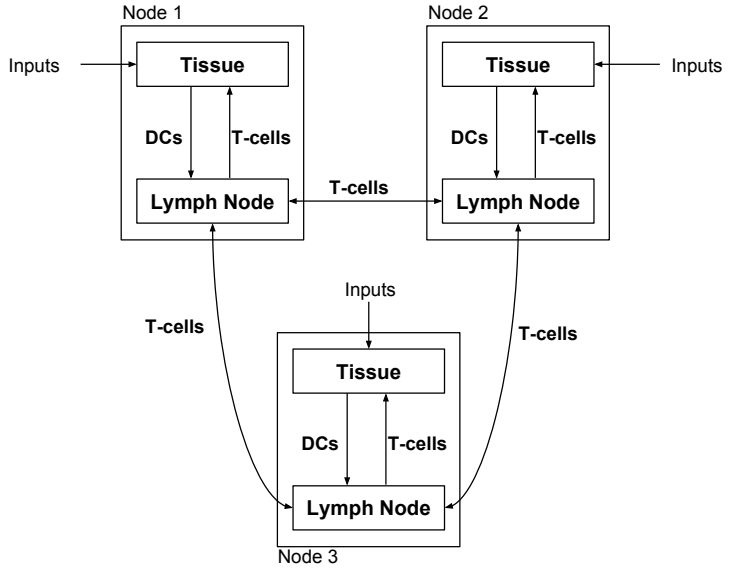


Figure 5.1 – Overview of CARDINAL-Vanilla.

5.3.1 Engineered System Overview

The architecture reads inputs from an input stream and classifies those inputs with a primary classifier. If the input can be classified, a response is actioned and a signature module is created for that input with a mapping to the response action. The signature modules are distributed to the other nodes of the networked architecture. More common inputs that present a threat retain a higher probability for selection and transmission to other architecture nodes. The other architecture nodes use the received signature module to respond to its matching input. These core behaviours provide self-hardening of the networked devices. Signature module prioritisation and validation refinement components guide the dispatch and the secondary-level classification. An engineered view of the architecture components is illustrated in Figure 5.2.

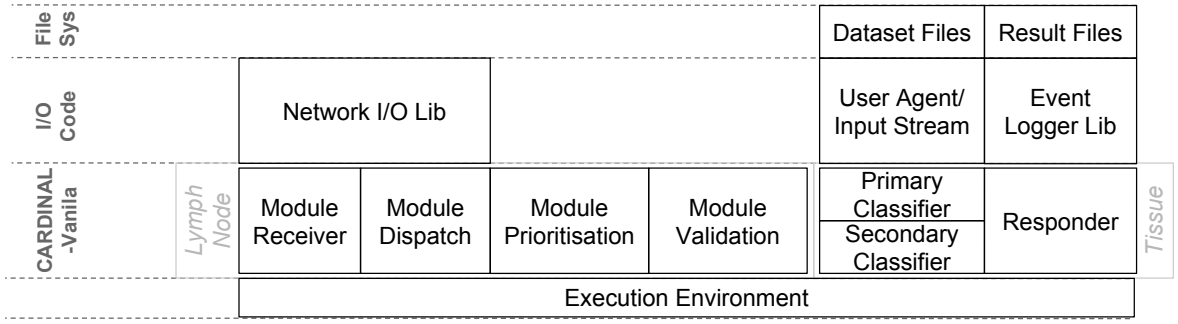


Figure 5.2 – Engineered System Overview of CARDINAL-Vanilla architecture client per node.

Term	Immunological Meaning	Computational Meaning	Structure	Usage
Antigen	A molecule recognized by an antibody or T-cell receptor	Input data source attributes	Data	
Danger signal	A molecule produced by cells undergoing stress/necrosis	Label classified as dangerous	Flag	Response mapping to *Type-I/II*
Safe signal	-	Label classified as safe	Flag	
Unknown signal	-	Unlabelled classification	Flag	
Dendritic Cell (DC)	A cell that presents processed antigen to T-cells	Primary classifier	Agent	Classify inputs
T-cell	A category of cells created in the thymus gland	Category of agent types	Category	
TCE	A category of cells that affect matching antigen	Category of responding agent types/ signature module	Category	
CTL (Mature)	A cell that causes apoptosis in infected cells	Agent/ signature module for *Type-I* responses (see 5.7)	Agent	Match & respond to inputs
Th2 (Mature)	A cell that affects responses to extracellular parasites	Agent/ signature module for *Type-II* responses (see 5.7)	Agent	Match & respond to inputs
Th1 (Regulatory)	A cell that assists responses by CTL cells	Agent container for unknown inputs	Agent	Assist priority of *Type-I* signature modules
TCN (Naïve)	A cell that can respond to novel antigen	Agent container for yet validated inputs (temporary)	Agent	Used in input validation process
Tissue	A region within which cells interact with antigen	Container for agents	Container	Facilitate monitoring & responding
Lymph node	A region within which T-cells and DCs interact	Container for agents	Container	Facilitate validation & distribution of agents
Lymphatic vessel	A conduit connecting lymph nodes	Communication stream between nodes	Container	Facilitate distribution of agents
Cytokine signal	A molecule that causes a change in cell behaviour	Internal agent communication signal	Message	Signal the danger category to TCN

Table 5.1 – Table of important terms, their summarised definitions from biological and computational perspectives, their structure representation and usage within the architecture. A complete list of parameter mappings are specified in Table 5.2.

5.3.2 Immune System Inspiration Overview

Kim et al.'s model drew upon components of the acquired and innate biological immune system and based its core methodology on danger theory (Matzinger, 1994). Table 5.1 provides a summarised set of terms and their meanings to narrate this biological description. Relevant sections within the biology background chapter are T-cells in section 3.3.2, dendritic cells in 3.3.3, signalling between cells in 3.4.1 and co-stimulation between cells and their thresholds in 3.4.2 and to frame the tissue defence see section 3.2.

5.3.2.1 Cell and Cytokine Behaviours

The behaviour modelled cells are the dendritic cells (DCs) and T-cells. The DC behaviours include collecting and presenting antigen to T-cells, migrations to the lymph nodes and localised release of interleukin cytokines IL4 and IL12 and costimulatory signals. Among the T-cell behaviours are migrations, maturation from naïve to effector cells, differentiation to three types of matured T-cells, activation and responses to matching antigen epitopes.

The tissue, periphery and the lymph nodes are the compartments within which these behaviours occur. The naïve T-cells (TCN) receive cytokines from the antigen carrying DCs, which in turn cause the naïve T-cell to mature. The modelled matured T-cells include the $CD8^+$ Cytotoxic T-Lymphocyte (CTL) cells and $CD4^+$ helper 2 T-cells (Th2) for their detect and respond capabilities, and helper 1 regulatory T-cells (Th1) to assist activation of CTL cells. Each of these immune system components are described in Chapter 3, for detailed reference see (Murphy *et al.* , 2012).

5.3.3 Agent Architecture Overview

The CARDINAL-Vanilla architecture is like many other agent-based systems, in that each *environment* conditionally determines an *agent*'s role and behaviour. Each agent has signal-based interaction behaviours and can *act* in an environment. There are multiple types of agents in an instance of the architecture executing on a single computer. The architecture (application) software is distributed across a network and thus this is a distributed multi-agent system. Figure 5.3 illustrates of the processes and agent behaviours operating in a single architecture node.

5.3.3.1 Actors and Agents

Every *actor* in the architecture is an agent. The agents are dendritic cells (DCs) and generalised T-cells. The T-cells have three subtypes, naïve (TCN), regulatory helper (Th1) and effector T-cells (TCE). The TCEs have two further subtypes CTL and Th2; these are the distributed responding agents. Each agent carries a signature; only the DCs and TCEs carry a mapping to a response.

5.3.3.2 Agent Interactions Summary

Several interaction events occur in each environment. Two-way signalling between TCEs with a matching signature will increase the number of agent clones, which we will call *priority*. Whereas TCEs with unmatched signatures will be de-prioritised at the time of an interaction event. The TCEs' priority will affect the signature's distribution and thus, response probability. One-way signalling from DCs

to TCNs occur and depend on the amount of danger the DC has experienced. Either a IL4, IL12 or costimulation signal will be released depending on the amount of the danger signal.

5.3.3.3 Three Agent Environments

The agents can be located in two spatial environments per computer; these are the *tissue* and *lymph node*. Each agent's transition between environments is conditional.

The tissue environment is the location where TCEs and a classifier to interact with the input streams. An input consists of an antigen and a signal, the attributes and its classified evaluation. Agents here classify and action responses to inputs depending on the agent type's response mapping. For example, the input's signal may dictate a specific response, alternatively the antigen may dictate the response. In danger theory-based Artificial Immune System (AIS) applications, antigen and signals are often mapped to input data and a decision based upon the input data. In a SCADA network security example we might map the antigen to control packet data sent to the microcontroller and the signal to an evaluation of the operational state of the monitored control systems.

The lymph node environment houses agent-to-agent interactions and agent-to-input interactions where inputs are carried by DCs. Here T-cell agents are prioritised or de-prioritised depending on whether matching signatures are carried. Input carrying DCs will be removed after a quantity of interactions. This is designed to let recent inputs have a larger effect upon the current internal priority state and let old inputs have no effect.

From the node-centric viewpoint, the final environment is the lymph node of another network-connected node. TCEs arriving will further prioritise or de-prioritise local TCEs. Novel TCEs will move to the local tissue to enable them to respond.

5.4 Processes of CARDINAL-Vanilla

The architecture model can be described by its processes. Figure 5.3 on page 65 illustrates the processes within the model. These are numbered as:

(0) Input Data Enters Tissue: Input data and an input label are read into an input buffer within the tissue component. They are referred to as antigen and signal. The label is converted into a numeric signal representation, specified in Eq.5.1.

(1a) Signal Collection: A DC collects the current signal, associated to the current antigen. If the signal is known and dangerous, the DC will collect the antigen (1c), respond (1b) followed by migrate to the tissue (2). These behaviours are described in 5.7.1 as Primary Classifier.

(1b) Effector T-cell and Dendritic Cell Response: The cell will respond to matching antigen. The TCE response is described in section 5.7.2 as Secondary Classifier and the DC response is in 5.7.1 as Primary Classifier.

(1c) Antigen Collection: The DC collects the antigen associated to the known danger signal.

(2) Dendritic Cell Migration from Tissue to Lymph Node: DC carries antigen and signal to lymph node. Described in section 5.8 as Module Validation or cell differentiation.

(3) Dendritic Cell Activates Naïve T-cell (TCN): The DC carrying the antigen and signal will activate (create) a TCN. If a matching TCE exists, the DC will interact instead (5).

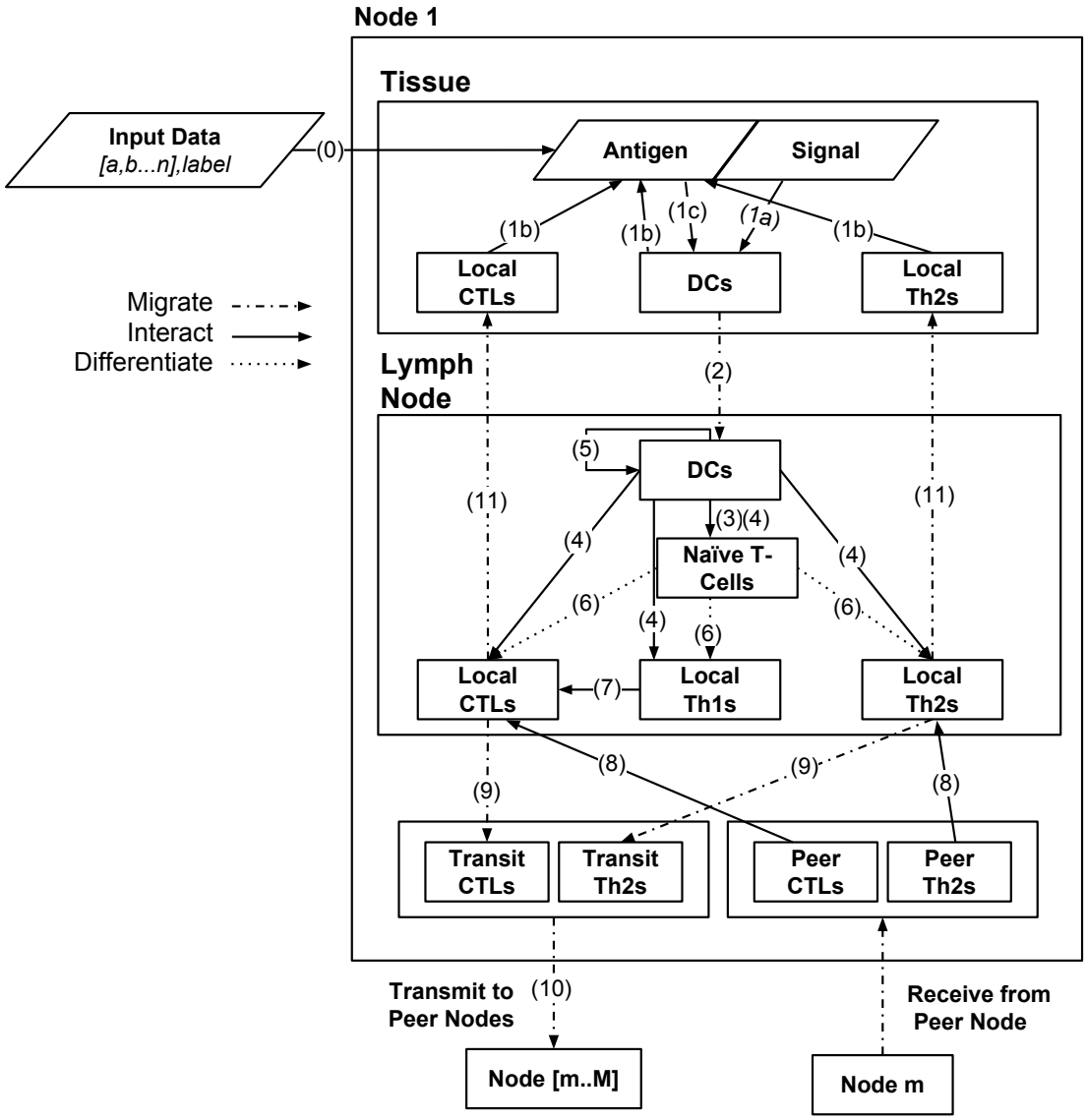


Figure 5.3 – Tissue and Lymph Node processes in CARDINAL-Vanilla in a single node. Numbered processes are described in 5.4. Diagram illustrates input data, data representation, agents, environments, agent interaction types (indicated by arrow line types) and agent network transmission messages. Diagram formatted as Fig. 2 from *(Kim et al., 2005)* for comparisons.

(4) Dendritic Cell Interacts with T-cells in Lymph Node: DCs interacts with T-cells as described in section 5.11.3 and in Eq.5.13, 5.14, 5.15.

(5) Dendritic Cells (DCs) Interact with DCs in Lymph Node: DCs interact with DCs. This enables the capability to reclassify a DC or Th1 agent that carries an input with an unknown signal. This will prioritise agents with common input signatures. The interaction is defined in Eq.5.16, 5.17. The lifespan decay process, upon each interaction, is described in Eq.5.15 and in section 5.11.3.

(6) Naïve T-cell Differentiates into Mature T-cell: TCNs differentiate into mature TCEs or Th1s. Described in section 5.8 as Module Validation Process.

(7) Regulatory Th1 cells Interacts with CTL cells: Th1 agents will help regulate the population (priority) of matching CTL agents, via an interaction. This process is defined in section 5.11.2 and Eq. 5.11, 5.12.

(8) Peer Effector T-cells (TCE) Interact with Local TCE Pool This process follows the explicit four stage procedure described in (Kim *et al.* , 2005)[p10]. Previously unseen Th2 or CTL cells are added directly into the local TCE pool. Received TCE (Th2 and CTL cells) interact with local TCE cells, leading to affected population (prioritisation). Described in section 5.11.1 as Module Prioritisation & De-Prioritisation. Equations 5.8, 5.9 and 5.10 describe these interactions.

(9) Effector T-cells (TCE) Migrate around Network: A number of Th2 and CTL cell are selected for distribution to other architecture nodes. The quantity depends on a local measure of accumulated danger, as specified in Eq.5.6, 5.7. The module selection process is described in section 5.10.3 as Module Selection for Transmission. Kim's model under-specifies the accumulated danger measure calculation and quantities, this affects the transmission selection mechanisms.

(10) Network Node Destination Selection: The selected Th2 and CTL cells are dispatched to other architecture nodes. The quantity of destination nodes depends on a local measure of accumulated danger, as specified in Eq.5.6, 5.7. The destination node selection process is described in section 5.10.2 as Destination Selection.

(11) Module Migration from Lymph Node to Tissue: Migrate new TCEs to tissue. Described in section 5.8.2 as Module Migration to Tissue. These TCEs provide storage of a response mapping; even in the event of changing input labels, i.e. under classification testing or as the input label classifier changes over time.

5.4.1 Process Differences

There are differences between our and Kim's process models. Firstly, the periphery is removed as it appears to describe the tissue. The cells have changed from data items to processes, as they have complex behaviours. Only the CTLs and Th2 cell agents are transmitted to other network nodes. The Th1 cell agents are not transmitted to enable a distinction and preference of local CTL agents over peer CTL agents, i.e. locally created detectors over network received detectors. The number of DC cell agent interactions have increased to enable reclassification of inputs with uncertain (or unknown) labels. DC cell agents now have a response pathway upon receipt of danger signals, which serves as an immediate response. The Kim's T-cell only response implies a delayed response. Among immunologists

it is well known that dendritic cells possess the capability to recognise bacteria, using toll-like receptors, and respond, described in 3.3.3.

5.5 Flow Control of CARDINAL-Vanilla

Figure 5.4 and Figure 5.5 on page 68 illustrates the flow control between Tissue and Lymph Node processes within the CARDINAL-Vanilla architecture model. Processes are described in section 5.4. The lettered decisions in the flow diagrams are:

<a> **Antigen Matches an Effector T-cell Agent Signature:** If this decision is passed, the TCE will respond following process (1c). Otherwise, the signal will be collected (1a) and evaluated in decision <b>.

<b> **Signal is Normal or Non-normal:** If the signal is normal or safe, the input will be discarded. Otherwise, the DC will attempt a mapped response (1b) followed by collecting the antigen (1c) and forwarding the input to the lymph node (2).

<c> **Naïve T-cell agent is Validated:** If the TCN has reached a validation (differentiation) threshold, forward to the differentiation process (6). Otherwise, continue to the next interaction process between Th1 and CTL agents (7).

<d> **Differentiates into an Effector T-cell Agent:** If this decision is passed, then the TCE will be forwarded to the Tissue (11), before continuing the next interaction process between Th1 and CTL agents (7). Otherwise, continue immediately to the next interaction process (7).

5.5.1 Flow Control Differences

There are flow control differences between the two models, as the diagrams depict. Concurrent execution is added for two reasons. Firstly, serial execution of networked applications and complex real-time applications are inappropriate or infeasible. Connections to many nodes and receipt of transmissions is simpler to coordinate when a thread is listening persistently. Transmissions can cause IO delays and therefore a single thread of execution would temporarily render the input processing in a static state. The real-time monitor flow is separated and minimised as real-time systems must minimise the delay between processed inputs. The Cell Storage components are added to share data between execution threads using lockable message queues, as is common in modern concurrency implementations. The main decision nodes are added to indicate conditional paths between processes. Others decisions exist and are described within the text.

5.6 Network Communications

The flow of biological T-lymphocyte and dendritic cells to key parts of the body is a concept drawn upon by Kim et al.'s model and shared by our view of the architecture. The transit of T-lymphocyte immune cells along lymphatic vessels to other lymph node organs is directly relevant to the networking and agent interaction design. Throughout transit, the cells experience interactions with other cells and

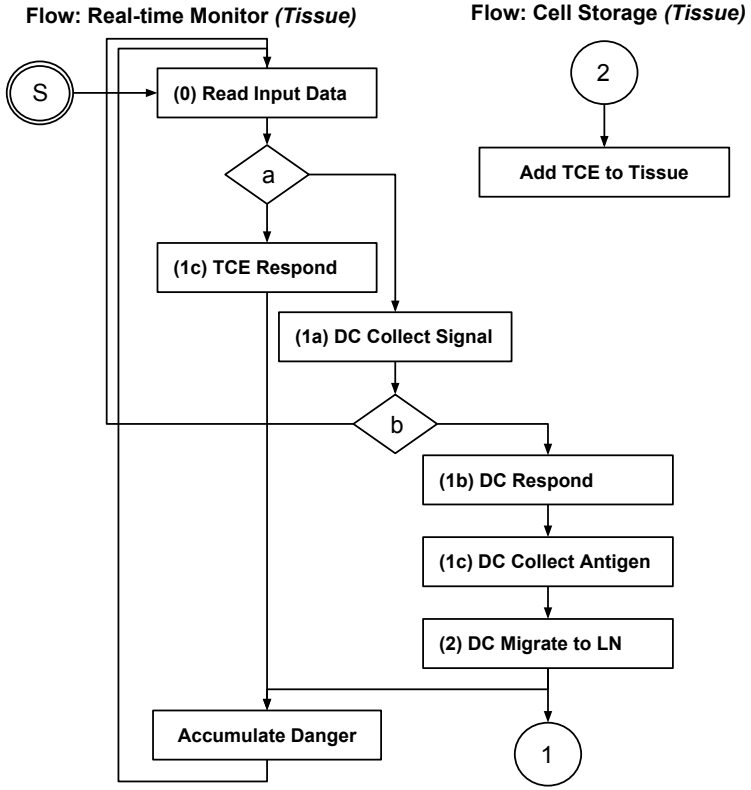


Figure 5.4 – Flow control diagram of the main Tissue processes.

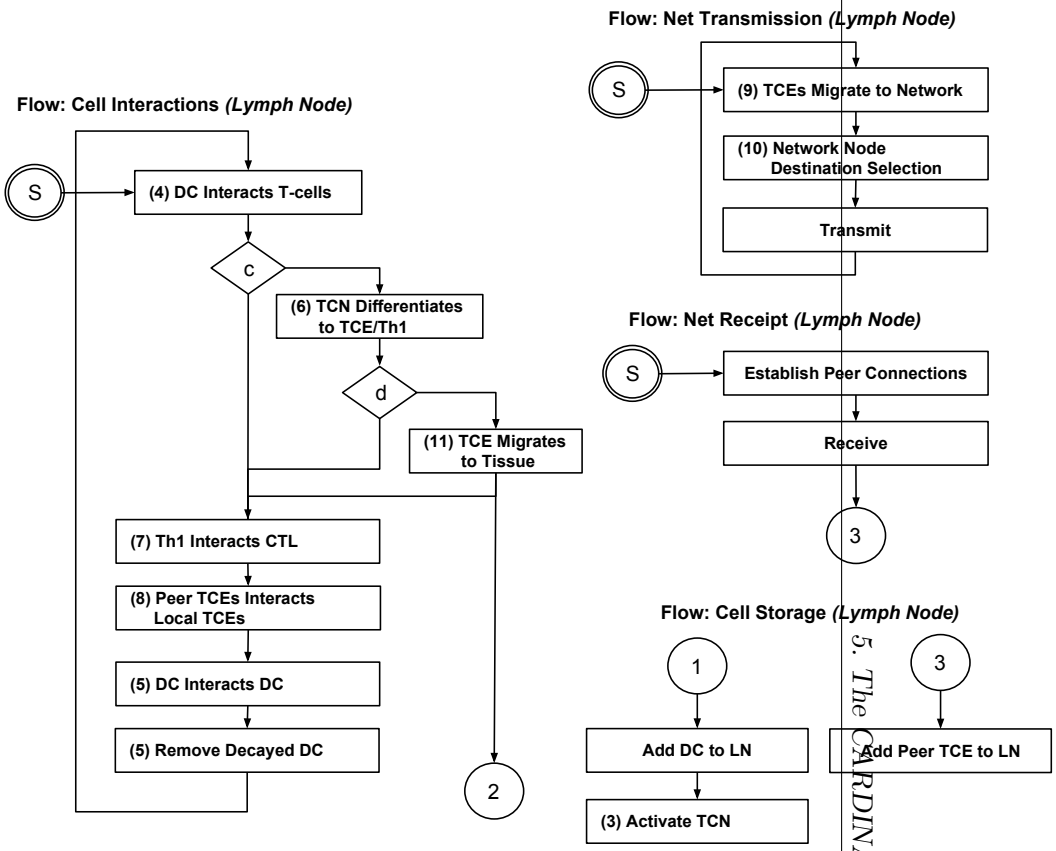


Figure 5.5 – Flow control diagram of the main Lymph Node processes.

The diagrams show the main flow control within the CARDINAL-Vanilla model. The diagrams are formatted as Fig. 3 in (Kim *et al.* , 2005) to aid comparisons. The key difference is the vital separation of execution paths. The execution steps are additionally more detailed.

Lettered (diamond shaped) decisions described in 5.5. Numbered processes are described in 5.4.

protein messengers. Intuitively, we interpret that the more clones a cell has will lead to its higher probability of being in the flow.

This approach leads to an heuristic which we specify in 5.11 and also leads to designs for the network structure, network protocol and the module dispatch decision making algorithms.

5.6.1 Network Connectivity & Decision Control

The network structure of the architecture is connected peer-to-peer. Each node maintains $n-1$ two-way unicast communication sessions. Each transmission's destination will vary depending on the internal state, e.g. priorities of the signature modules and others. The architecture uses decentralised decision making. Each node's internal state dictates its local actions and communications to other nodes. No omnipresent actor directs the network-wide behaviour. Local decisions taken by a node depend on the recent local inputs observed, the internal model of cells (agents), its signatures and its own validated definition of danger.

5.6.2 Network Protocol

One application message protocol is used to transmit detector & response modules, while other messages oversee the experimentation run of the distributed system. We use the Java language's built-in serialisation and marshalling of objects on a stream and the received object's class name is used to trigger an associated action. The internet protocol (IP) and transmission control protocol (TCP) provide the network node and application addressing.

The module transmission message type is a data object containing a set of selected and previously validated TCE agents. Upon arrival, these TCEs interact with local TCEs and new TCEs are directly added into the local lymph node component, in the manner stated in 5.11.1.

The most important experiment message type is a *STOP* marker to inform another node that the sender's experiment activities have finished. When each node has received this message from all other nodes, they can each prepare to terminate the experiment run. This avoids an experiment run reaching an irrecoverable state caused by unhandled asynchronous termination.

5.7 Classification and Responses

The architecture requires that a classification algorithm and response scripts be integrated. For a Supervisory Control and Data Acquisition (SCADA) self-healing system selecting, configuring and training the right classifier model(s) and choosing the right mapped response actions to those classifications are important tasks to investigate. However, classifier evaluation in this thesis is not our focus. As such, we have implemented the Occam's razor of lifelong or online machine learning non-time-series classifiers. In summary, the primary classifier depends on labelled inputs to establish a response mapping and the secondary classifier depends on previously learnt inputs and response mappings. Invoked responses log the response mapping category to a result file.

The classifier maps a danger input to either *Type I* or *Type II* responses, represented by the CTL and Th2 agents respectively. The graded classification will distinguish inputs as normal, unknown or unconfirmed, and the two categories of known danger. A threshold or label can distinguish *Type-I*

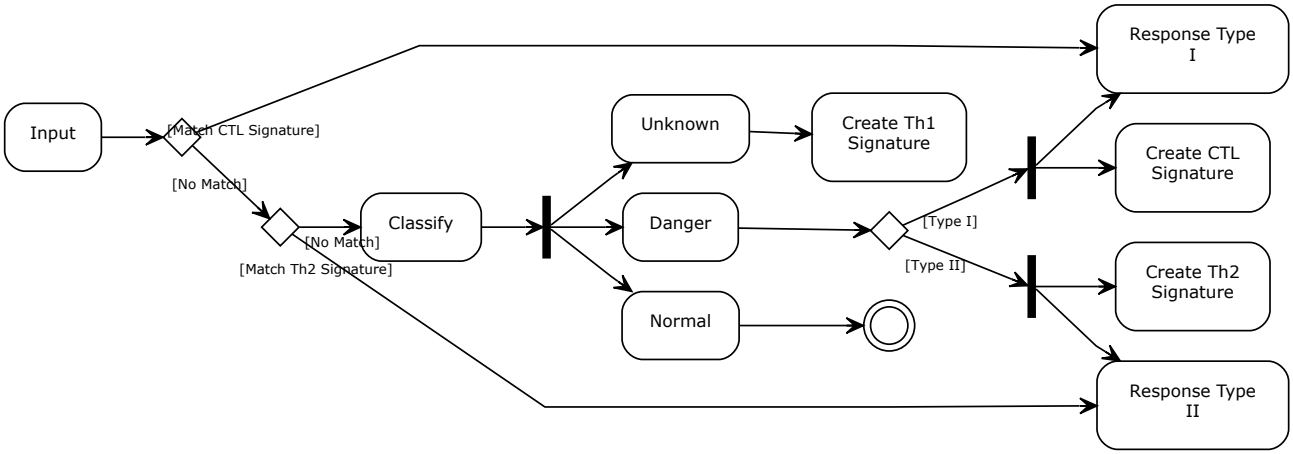


Figure 5.6 – Summarised view of Input to Response from the classifier decision perspective.

from *Type-II* dangers; however, this work uses the *Type-I* danger category only. Each new non-normal input will undergo the validation process, as we define in 5.8. The validated input will become a distributable signature module and be subject to prioritisation in 5.11. The module forms the self-hardened, secondary-level of response at the local monitoring component, as described in 5.7.2.

5.7.1 Primary Classifier

The primary classifier assesses an input to produce a mapping to a response. The experiment chapters use a biased supervised classifier to label inputs as *normal* or *attack*. In line with the danger theory approach the normal signals, S_S are ignored and discarded. All attack labels are grouped as *Type I* danger signals, S_D such that only CTL agents are under evaluation. Therefore attack labelled inputs are mapped to the *Type I* response. Note that Th2 agents have identical agent interaction profiles to CTL agents, but have a different response mapping. Test inputs, i.e. without signal labels, that do not match a known signature are considered as unknown signals, S_U. The classifier assigns a signal value for each label, as follows:

$$S = \begin{cases} S_S = 0 & , \quad if \;\; label = normal \\ S_D = 1 & , \quad if \;\; label = attack \quad e.g. \;\; Type\text{-}I \\ S_U = * & , \quad if \;\; label = none \end{cases} \tag{5.1}$$

Upon classification of a danger signal input, a DC agent will execute the mapped response. Then, if the input is classified as non-normal, the DC will carry it to the lymph node in order to begin the T-cell validation process.

Kim et al. envisaged the integrated primary classifier to be the dendritic cell algorithm (DCA) classifier (Greensmith *et al.* , 2006). However, Stibor et al. have shown an alternative to DCA is required if the application problem is not linearly separable (Stibor *et al.* , 2009), with further evaluations in (Gu *et al.* , 2011). Our work has evaluated two replacement classifier pipelines in (Song *et al.* , 2013) and (Song *et al.* , 2014) which report good accuracy on network security datasets. However neither pipeline use online data stream learners; as such a problem-domain specific classifier with online learning is

needed for future applications of the architecture.

5.7.2 Secondary Classifier

The secondary classifier calculates a hashing function value $H(x)$ for the antigen A_n component of a new input. The hash value is matched against a table of existing signature hashed values. Therefore, its accuracy is dependent on the primary classifier (5.7.1) and the module validation procedure (5.8).

Hashing functions reduce the attribute set dimensionality with the feature extraction methodology; losing the original semantics of the input features while keeping the separability (Liu & Motoda, 1998). As we know, hash function lookups have a time complexity of O(1) and a space complexity of O(n) on average, where n is a reduced byte size to the original A_n value. A specific hash function is limited by its maximum unique mapped values. Generally speaking, a hashing function is a fast string comparison approach that translates the literal characters into a numerical space in order to numerically evaluate the comparison, as Equation 5.2. The growth in space complexity and it's entirely biased learning model ensure its suitability to experimental cases and to strict decision making cases where the potential learning space is finite and small.

$$H(A_n) = \forall x \in A_n(h(x)) \tag{5.2}$$

Each existing signature has a response mapping, as specified by the primary classifier and validated by the danger input validation process. CTL and Th2 agents carry a response mapping and upon a match will initiate the mapped response. When the response is issued, the matched TCE agent will increment its RS_{TCE} value; thus affecting its heuristic priority value.

5.7.3 Responses

In the CARDINAL-Vanilla experimentation we are interested to know when and under what circumstance a response is issued. Kim et al. specified "weak" and "strong" responses, associated to Th2 and CTL agent types. In CARDINAL-Vanilla, a log entry is reported with the response type, i.e. *Type-I/II*, the triggering input and the responding cell type, i.e. Th2 or CTL.

In future, applied *Type I* and *Type II* responses should be problem-domain specific. Appropriate responses for the SCADA control network environment can be as described in C.1. These include blocking port numbers in a firewall, alerting an administrator, issuing a command to stop the controller operation, among others.

5.8 Module Creation & Validation

The signature module creation is the result of the validation process, which Kim et al. and immunologists refer to as cell *differentiation*. The cell changing from a naïve to a mature state. Between a new input and the validation process are two intermediate steps, as described by Kim et al. First a DC agent will evaluate the signal and choose to respond or not, as redefined in section 5.7.1. The DC agent will then migrate to the lymph node. A new TCN agent will be created with the input information. The DC and the new TCN agents will then join the pool of DCs and T-cells. The validation process, or differentiation in Kim's terms, will cause that TCN to become a TCE or Th1 agent.

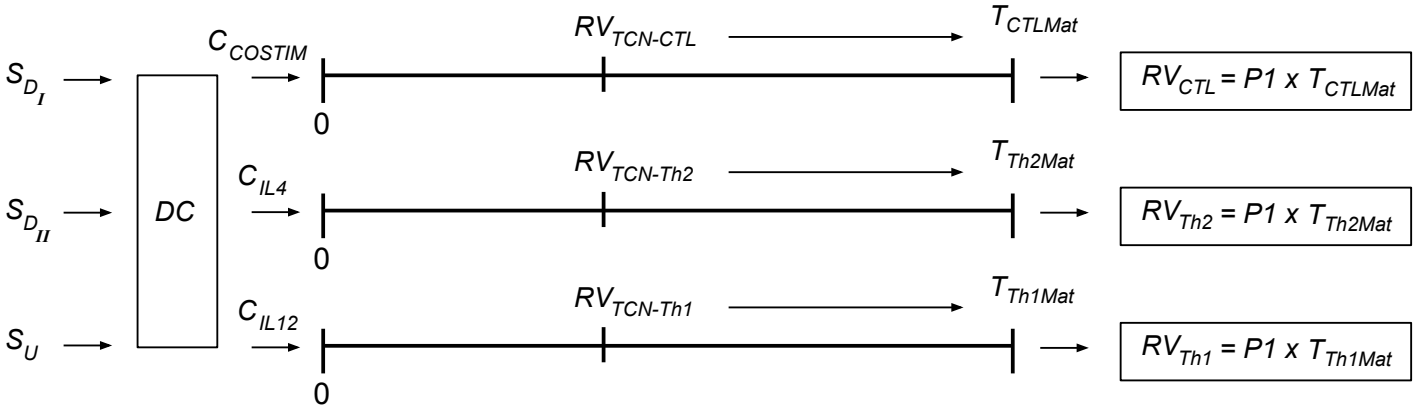


Figure 5.7 – Danger Input Validation Process - Naïve T-cell Maturation and Differentiation with DC Interaction

The module validation process validates an input carrying TCN agent when its associated sum of danger exceeds a threshold. Kim et al. defined this threshold as a "suitable period of time" in their worm malware case. We have respecified this as a sum of danger, accumulated over one or more inputs with a matching antigen. Figure 5.7 illustrates the behaviour. It shows the process is dependent upon each matching input's danger signal value given by the primary classifier. We will describe the process next.

5.8.1 Module Validation Process

A DC agent will carry non-normal inputs and their signal value from (S_D or S_U) the tissue environment to the lymph node environment. In the lymph node, the DC will transfer the antigen signature and release a cytokine signal C_* to a newly created naïve T-cell (TCN) agent. The cytokine signal increases the corresponding runtime priority value RV_{TCN*} toward one of the three differentiation thresholds. The first threshold to be exceeded will validate the input and create the permanent differentiated T-cell; either CTL, Th2 or Th1. A later arriving DC carrying a similar or equal antigen will release a cytokine signal C_* associated to its danger signal. This will invoke the same maturation process as with TCNs, as shown in Figure 5.7.

The differentiated T-cell agent will have an initial runtime priority RV_* value equal to $P1 * T_{*Mat}$ and enter the local population of TCEs and Th1 agents in order to be subjected to interaction behaviours. The $P1$ parameter affects the initial quantity of T-cell agent clones upon differentiation and thus affects a signature module's starting heuristic priority value. A differentiated T-cell agent is a validated signature module.

5.8.2 Module Migration to Tissue Process

When a TCE agent has been validated it will migrate to the tissue environment. This ensures known validated signatures are capable of responding to inputs as part of the secondary classifier in 5.7.2. The non-responder Th1 agents remain only in the lymph node. In terms of implementation, a linked reference to a TCE agent is created and sent to the tissue. A linked reference causes the validated TCE's priority RV_* value to be affected by peer, local and antigen input interactions; where a *true* migration, i.e. move, or a copy would not.

In Kim et al.'s model, only "effector cells with a positive clone value [will migrate]". If we chose to use the priority value RV_{TCE} heuristic mechanism as the deciding factor for this migration, the effective response would be reduced. This is bad for small use cases, but good for scalability. However, this depends on whether the priority heuristic is precise.

5.8.3 Choosing Thresholds and Cytokine Increase Values

The threshold and increase values affect the likelihood of differentiation. To equalise the values per cell line will lead to a likelihood decision based on the dataset input order. To incorporate the signal value will cause a sliding affect upon *Type I* danger, *Type II* danger and unknown signals, assuming $S_U < S_{D_{II}} < S_{D_I}$.

In our experimentation, the primary classifier mapped a signal value of *Type I* danger to 1 and unknown signal to $*$, no value. We opted to equalise the rate with equal increment values of 1 and no addition of the input's danger value. The CTL differentiation threshold was set at 1, which removes the validation step for training inputs labelled as dangerous. The Th1 differentiation threshold was set at 2. This was done to leave the possibility of relabelling the unknown input, via the DC to DC agent and via the DC to TCN agent interaction process.

5.9 Time Scale: Moving Time Window

The statically-sized moving time window enables us to track behaviour over a fixed period of passed time. It is used by two processes. First, is for the Volume Selection decision, described in 5.10.1, to track danger quantities over time using the accumulated sickness value, S_v. The second aids the heuristic priority decision. Kim et al. define an accumulation of danger over two time steps. Our change here is to define that period using a static moving window of length $P3$, where each moving block of that window is measured in 500 millisecond intervals. This enables us to deal with real-time behaviours and beyond a stream of dataset inputs, as the time step approach implies.

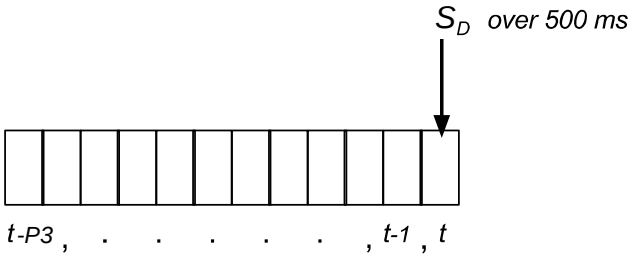


Figure 5.8 – Diagram of the data structure representation for the moving window of accumulated danger. $t-P3$ is the earliest time block and next to be removed, t is the current and most recently added time block.

The sum of danger values S_v over the length of the time window, is defined by Equation 5.3. i is the index to window blocks and $\sum S_{D_i}$ is the sum of danger in the i-th block. We have selected 500 milliseconds as the duration of a time block. Therefore, the i-th block contains the accumulated danger values between $t-(i*500\text{ms})$ and $t-((i+1)*500\text{ms})$ from the current time t. $P3$ is the quantity

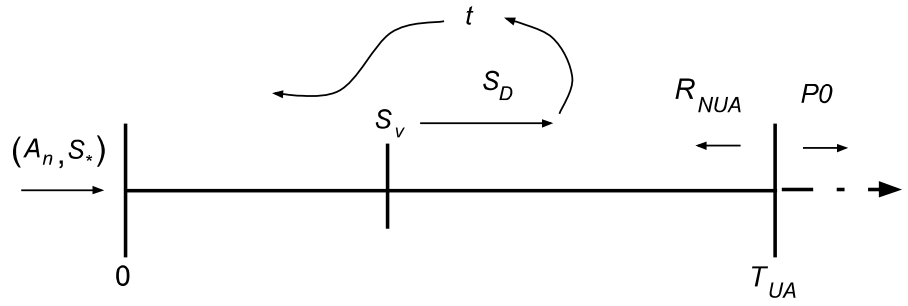


Figure 5.9 – Volume Selection Feedback Loop - Sickness Focus. T_{UA} is the under attack threshold. $P0$ is the under attack percentage volume to transmit. R_{NUA} is the percentage to send while not under attack. S_v is the recent danger level affected by damaging inputs. S_v is increased by S_d danger signal inputs, otherwise its value will decay over time. A_n is the input antigen or data input values, while S_* is the input data's label.

of windows over which S_v is accumulated and thus is a threshold modifier for the Volume Selection decision.

$$S_v = \sum_{i=0}^{P3} \sum S_{D_i} \tag{5.3}$$

The second process that uses the moving time window, has two instances within each TCE agent. These track the recent number of responses RS_{TCE} and the recent priority value RV_{TCE} of the TCE. They aid the decision to increase or reduce the priority leading to a change in the probability of its selection for transmission to other nodes. These priority modifier decisions are shown in Equation 5.9 and Equation 5.8.

The sums of S_{RS} and S_{RV} are calculated similarly to S_v. Equation 5.4 and Equation 5.5 show the semantic difference is only in the variable value accumulated.

$$S_{RS} = \sum_{i=0}^{P3} \sum RS_{TCE_i} \tag{5.4}$$

$$S_{RV} = \sum_{i=0}^{P3} \sum RV_{TCE_i} \tag{5.5}$$

5.10 Module Dispatch Decisions

5.10.1 Volume Selection Feedback Loop

The volume selection feedback loop shown in Figure 5.9 takes inputs from the monitoring sensor and determines if the system is under attack. If so, the next transmission will uprate the percentage volume of destinations, as in 5.10.2 and of modules sent, described in 5.10.3.

Each sensor input consists of an antigen (data input values) and a danger signal value (data input's label), of which the signal inputs with a non-zero value have an affect on the loop. Each input danger signal value, S_D, will cause the accumulated danger value or sickness value, S_v to be increased. Equation 5.3 defines S_v. Over t time the accumulated S_v value will be decreased, as described in 5.9. When

the value of S_v breaches the T_{UA} decision threshold the behaviour is modified to send a volume at the under attack rate $P0$, otherwise the not-under-attack rate R_{NUA} volume is used:

$$S_v > T_{UA} \rightarrow P0 \tag{5.6}$$

$$S_v \leq T_{UA} \rightarrow R_{NUA} \tag{5.7}$$

The other parameters affecting this decision and time-dependent feedback loop are as follows. S_D is the value of danger for a given input. S_v is the accumulated danger value, explicitly defined in Equation 5.3. In 5.9, we discuss the S_v variable and the static moving time window mechanism used to track S_v.

In the dataset validation cases, we set the value of S_D to be 1, and 0 for non-danger inputs. For other uses, it is recommended that the value of S_D be the danger score calculated by the domain-specific classifier. The default values and ranges for each of the parameters can be found in Table 5.2.

5.10.2 Destination Selection

This decision is initiated upon each transmission to give a set of neighbouring destination nodes to which the current transmission will be sent. The decision inputs are the set of node addresses $n_1, ..n_N \in \mathbf{N}$, the random number generator RNG_1, its seed S_1 and the transmission volume percentage v, given in 5.10.1. v is converted into its respective ceiling quantity $q_N = \lceil v{*}N \rceil$ of destination nodes. Then $N - q_N$ addresses are removed from $\mathbf{N}$ with sequential uniform probability, to give the set of Q_N destinations for this transmission.

5.10.3 Module Selection for Transmission

This decision selects a set of modules to send to the Q_N destinations. Similarly to the destination decision, the inputs are the random number generator RNG_1 and its seed S_1, the transmission volume percentage v and the set of modules $m_1, ..m_M \in \mathbf{M}$, including both CTL and Th2 agents. The volume quantity becomes $q_M = \lceil v * M \rceil$, with which q_M modules are to be selected for this transmission. The individual modules are selected with roulette wheel probability selection using each agent's priority RV_{TCE} as a fitness value, where higher is best. We describe the RV priority and heuristic in 5.11. The set of Q_M selected modules are then sent to the Q_N destinations.

5.11 Priority Heuristic for Signature Module Distribution

The priority value RV provides the key heuristic to measure the importance of an input signature. The heuristic states that if the agent carrying the signature responded on more occasions than its RV value, then increase its priority. Its use in the architecture affects signature probability selection for transmission to the other nodes. The priority is increased and decreased in the following ways.

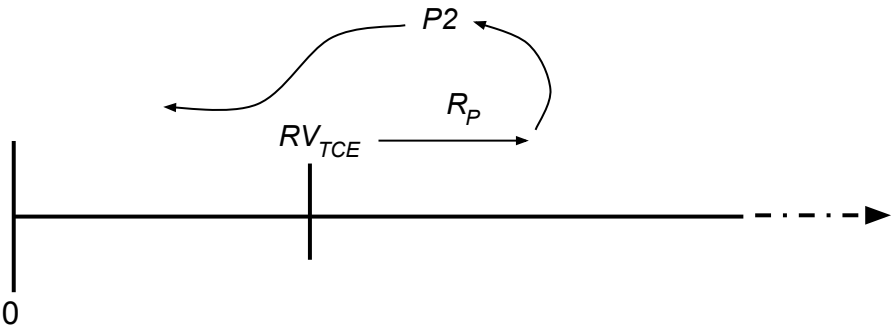


Figure 5.10 – Effector T-cell Feedback Loop upon Cell Interaction

5.11.1 Module Prioritisation & De-Prioritisation

The module prioritisation feedback loop shown in Figure 5.10 is part of each effector T-cell agent (TCE) and affects their priority runtime value, RV_{TCE}. This translates into an importance score for each TCE agent which determines each agent's probability of selection for transmission to other nodes. Finally, this enables other nodes to respond to its signature.

The following interactions occur between the local TCEs and the TCEs received via network transmission. Kim et al.'s model states that local Th1s will interact with received Th1s, in addition to TCEs interacting with TCEs. We have chosen to remove the Th1 transmissions and therefore interactions upon network receipt. This is done to improve the priority of signature modules (TCE agents) with locally received inputs. Section 5.11.2 describes how Th1 agents increase the priority values of CTL agents. In summary, Th1s store local input data with an unknown label; where CTLs store input data with a label from either the local node or a peer node. Therefore local Th1s will increase the priority values of local input-focused CTL agents over peer received CTL agents.

As we have described, each TCE's signature is composed of an antigen A_n and signal S_*. TCEs without matching antigen signatures are de-prioritised by $*P2$, the suppression rate. TCEs with matching antigen signatures are uprated by R_p, the proliferation rate if the following clause holds. If the local TCE's accumulated number of responses is more than its priority, then its priority value is increased. Otherwise, its priority is too high for its current use, we therefore reduce its priority. To avoid keeping duplicates, the matching received TCE is discarded after all interactions have completed. Next, we shall express these cases more explicitly in formal notation.

Prioritise Similar Local Effector T-cells via Received Peers

Equation 5.8 expresses the element-wise prioritisation case. Such that, for all a_{A_n} antigen in the local set of TCEs A_{TCE}, if a similarity exists with an antigen b_{A_n} in the received set of TCEs B_{TCE} and a's sum of recent responses $a_{S_{RS}}$ is greater than it's sum of recent priority $a_{S_{RV}}$, then increase the corresponding local TCE's priority by R_p. S_{RS} and S_{RV} calculations are defined in Equation 5.4 and Equation 5.5.

$$\forall a \in A_{TCE}((\exists b \in B_{TCE} : a_{A_n} \simeq b_{A_n} \wedge a_{S_{RS}} > a_{S_{RV}}) \rightarrow a_{RV_{TCE}} + R_p) \tag{5.8}$$

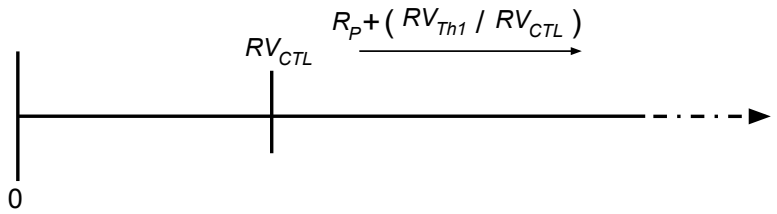


Figure 5.11 – Prioritisation of CTL Responders through Regulation.

Deprioritise Similar Local Effector T-cells via Received Peers

Equation 5.9 expresses the de-prioritisation case for matching antigen. Under the same matching antigen condition, when the sum of recent responses $a_{S_{RS}}$ is less than or equal to the sum of recent priority $a_{S_{RV}}$, then suppress the runtime priority value RV_{TCE} of a by $*P2$, as in:

$$\forall a \in A_{TCE}((\exists b \in B_{TCE} : a_{A_n} \simeq b_{A_n} \wedge a_{S_{RS}} \leq a_{S_{RV}}) \rightarrow a_{RV_{TCE}} * P2) \tag{5.9}$$

Deprioritise Dissimilar Local Effector T-cells via Received Peers

Equation 5.10 shows the de-prioritisation case for non-matches, which is the set-wise logical negation of the matching cases. These remaining local but unmatched TCEs are reduced by the suppression rate $*P2$. Therefore, if there is not a single case where an antigen c_{A_n} in the local set is similar to an antigen b_{A_n} in the received set, then suppress the runtime priority value RV_{TCE} attached to c with $*P2$, as:

$$\nexists c \in A_{TCE} : c_{A_n} \simeq b_{A_n} \wedge b \in B_{TCE} \rightarrow c_{RV_{TCE}} * P2 \tag{5.10}$$

These examples using TCE summarise both cases for CTL and Th2 agent interactions, including prioritisation and de-prioritisation. Explicitly, these comparisons occur between CTLs and CTLs, and between Th2s and Th2s. Our specific matching approach compares numerical hashed values of the categorical input data without the label and without the signal value; however a distance-based correlation could be envisaged for numerical variables as suggested by our use of mathematical notation for similarity. See Table 5.2 for parameter ranges and default values.

5.11.2 More Prioritisation: Through Regulation

The further prioritisation interaction shown in Figure 5.11 takes locally residing Th1 and CTL sets as inputs and is initiated on each scheduled interaction process. The interaction increases the selection probability for higher danger signatures, carried by CTL agents, that have similar antigen and differing levels of danger signals to those carried by Th1 agents. Matched signatures of CTL and Th1 agents will cause an uprate of the CTL's priority and the Th1's priority.

If an element-wise match holds then an increase occurs in the runtime priority value for the CTL, as shown in Equation 5.11, and in the Th1, as shown in Equation 5.12. For all antigen a_{A_n} from the

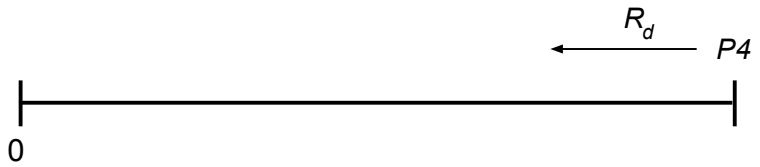


Figure 5.12 – DC Lifespan and Decay Rate

local set of CTL agents, denoted by $a \in A_{CTL}$, if there exists a match to an antigen b_{A_n} in the local set of Th1 agents, $b \in B_{Th1}$, then initiate the increase. R_p is the proliferation rate. $a_{RV_{CTL}}$ and $b_{RV_{Th1}}$ are the runtime priority values of agents that carry a matching antigen:

$$\forall a \in A_{CTL}((\exists b \in B_{Th1} : a_{A_n} \simeq b_{A_n}) \rightarrow a_{RV_{CTL}} + R_p + (b_{RV_{Th1}}/a_{RV_{CTL}}), \tag{5.11}$$

$$b_{RV_{Th1}} + R_p) \tag{5.12}$$

When the starting priority of the CTL is lower than the Th1, then a more extreme increase occurs. Unmatched signatures are unaffected in this interaction.

5.11.3 More Prioritisation: Dendritic Cells and Decay Rates

The dendritic cell (DC) agents will interact with naïve TCN agents and the differentiated CTL, Th2 and Th1 T-cell agents residing in the local lymph node.

Dendritic Cell Increases Differentiated T-cell Priority

Upon presentation of a matching antigen, a differentiated T-cell will proliferate, its priority will increase. This is shown in Equation 5.13, where A_n is the antigen signature of an agent, RV_{TCell} is the current priority value of the T-cell and R_p is the rate of proliferation. The local set of DCs $a \in A_{DC}$ interacts with the local set of all T-cells $b \in B_{TCell}$, as:

$$\forall a \in A_{DC}((\exists b \in B_{TCell} : a_{A_n} \simeq b_{A_n}) \wedge a_{S_D} \rightarrow b_{RV_{TCell}} + R_p) \tag{5.13}$$

Dendritic Cell Signal is Replaced

Where the matching DC has an unknown signal, S_U and the matched T-cell has a known danger signal S_D, the DC's signal will also be overwritten:

$$\forall a \in A_{DC}((\exists b \in B_{TCell} : a_{A_n} \simeq b_{A_n}) \wedge a_{S_U} \wedge b_{S_D} \rightarrow b_{RV_{TCell}} + R_p \ , \ a_{S_U} := b_{S_D}) \tag{5.14}$$

Dendritic Cell Lifespan Decays

Each DC's lifespan will decay per interaction regardless of whether or not the interaction led to a match. The remaining lifespan L_{DC} will decrease by R_d, the rate of decay, as shown in Figure 5.12 and expressed in Equation 5.15. Its lifespan is initialised at $P4$, when the lifespan value reaches zero the DC will be marked for deletion and removed when this scheduled interaction phase ends.

$$\forall a \in A_{DC}(\forall b \in B_{Tcell} \rightarrow a_{L_{DC}} - R_d) \tag{5.15}$$

Dendritic Cell Reclassifies Other Dendritic Cell Signal

The DC agents will also interact with other local DCs. This causes additional lifespan decay. Upon a match, if a danger signal is known and an unknown signal exists, the known signal will take precedence and replace the unknown. Equation 5.16 and Equation 5.17 show both change cases of this. In Equation 5.16, if the a DC has an unknown signal S_U and the a' DC has a known danger signal S_D, then the a' DC's signal will replace the a DC's signal. Vice-versa for the second case:

$$\forall a \in A_{DC}(\forall a' \in A'_{DC}/a : a_{A_n} \simeq a'_{A_n} \wedge a_{S_U} \wedge a'_{S_D} \rightarrow a_{S_U} := a'_{S_D}) \tag{5.16}$$

$$\forall a \in A_{DC}(\forall a' \in A'_{DC}/a : a_{A_n} \simeq a'_{A_n} \wedge a_{S_D} \wedge a'_{S_U} \rightarrow a'_{S_U} := a_{S_D}) \tag{5.17}$$

Dendritic Cell Reclassifies and Matures Naïve T-cell

The DC agents will also interact with local TCN agents that have a matching antigen. The purpose is to replace an unknown signal S_U of a TCN with a known danger S_D signal. This behaviour is defined by Equation 5.18 and is similar to the DC to DC interaction case:

$$\forall a \in A_{DC}(\forall b \in B_{TCN} : a_{A_n} \simeq b_{A_n} \wedge a_{S_D} \wedge b_{S_U} \rightarrow b_{S_U} := a_{S_D}) \tag{5.18}$$

Upon a match, the DC will additionally transmit a cytokine message to the TCN agent in a similar manner to Figure 5.7 and the procedure in section 5.8. If the DC holds a S_D, it will release an appropriate cytokine, either C_{IL4} or C_{COSTIM}. Otherwise, if the TCN holds a S_D, the DC will release a message to exacerbate the currently held signal. Where both the DC and TCN have unknowns signals, the TCN will receive a cytokine message matching its current strongest signal RV_{TCN*} value.

5.12 Parameters of CARDINAL-Vanilla

Table 5.2 shows a complete list of parameters with their default values and ranges. This section will describe the parameters and identify those that affect the key components of the architecture.

The key parameters chosen are labelled P0 to P4. Each impacts one or more of the key decisions or interactions, including the feedback loop, decay rates and thresholds. The experimentation will

investigate variations of these parameters. Below we will discuss each of these parameters and justify a range of test values. First we will look at the remaining parameters and variables listed.

5.12.1 Choosing the Key Parameters

First we can remove the variables as a result of equations, which are the initial priority value variables V_* and the runtime priority and response variables RV_*, RS_*, the calculated CTL-Th1 interaction increase quantity R_R and the sums S_*. Secondly we can remove the parameters with values of 0 or 1, which are the signal values given by our classifier S_*, the cytokine signal values C_* and then the rates of increase and decrease, which are the decay rate R_d and the proliferation or increase rate R_p. The proliferation rate would be worthy of investigation had we not included the two other components to the priority feedback loop, the starting priority $P1$ and suppression rate $P2$ modifiers, in our list of key parameters.

This leaves the more interesting parameters. The validation thresholds T_{*Mat} whose values were stated and justified in 5.8.3. The random number generator seed S_1 that enables the uniform randomness for the roulette wheel and probability selection mechanisms within the architecture; this is to be chosen within the experiment design. The final two are the under attack threshold T_{UA} and the not under attack volume transmission percentage R_{NUA}.

The T_{UA} threshold is an accumulated quantity threshold that is summed using the moving window structure. Variation of the $P3$ parameter modifies the length of the window and therefore the semantic of the threshold. Thus, while we do not modify the threshold value, we can evaluate the underlying threshold decision by modifying the value of the single $P3$ parameter. The threshold value of 50 was arbitrarily selected and based upon a "small" quantity of inputs relative to our datasets and a typical application case. The R_{NUA} volume percentage parameter counters the higher rate of flow parameter, $P0$.

R_{NUA} gives a small constant flow of data between the nodes when the system is not under attack; semantically the value should remain less than the value of $P0$. However, at one level, R_{NUA} semantically shares the same parameter space as $P0$. The value of 0.25 was arbitrarily selected and based upon the architecture's need to transmit modules to other nodes at a constant heartbeat-like flow rate. Lowering the value toward 0 is not intuitively interesting.

The remaining parameters are the key parameters, described below.

5.12.2 Under Attack Volume Percentage Parameter (P0)

This parameter is a percentage quantity of transmissions; it is the alternative and increased percentage used when there is evidence that the local system is under attack. The percentage is used to select the number of modules and destinations when the under attack condition has been met. Equation 5.6 defines the condition. The default value of $P0 = 0.75$ counterpoints the not under attack condition value, $R_{NUA} = 0.25$.

P0	
Default value:	0.75
Continuous Range:	$[0.25 - 1.0]$
Discrete Range:	$[0.25, 0.33, 0.5, 0.66, 0.75, 0.9, 1.0]$

The module selection uses roulette wheel selection. Thus, the chance of selecting the same module increases when some modules have high priority values and many have low priority values. The number of items selected increases with the $P0$ value. By design, this does not affect the destination selection decision. We can expect the reported quantity of data transmitted to increase as $P0$ increases.

5.12.3 Initial Priority Value Multiplier Parameter (P1)

This parameter will inflate the initial heuristic priority value for each signature module. This value is later increased by increments and decreased by division $P2$, according to the heuristic. The $P1$ parameter sets the initial size, that is countered by the $P2$ parameter.

P1	
Default value:	2
Continuous Range:	$[0.0 - 4.0]$
Discrete Range:	$[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]$

The heuristic decisions will decrease a module's priority value while it has no need to respond. This parameter is thus akin to a measure of time, over which the priority value for a new module can stay relatively high. When the parameter's value is higher, we expect it to maintain the buffered value for longer; when low, we expect new modules to reduce their priority quickly. The effect of this that newer modules get selected for transmission, thus the network has a stronger relevant resistance.

5.12.4 Priority Value Suppression Parameter (P2)

This parameter decreases a module's priority value by multiplication. The priority value is initially affected by $P1$ and increased by the rate of increase R_p. With the parameter value at 1.0 no decrease is present, closer to 0.0 will cause the priority value to decay very quickly. We include 1.0 in the range, to investigate the effect of no suppression on the priority values. > 1.0 is in the range to investigate a priority that only increases.

P2	
Default value:	0.95
Continuous Range:	$[0.0 - 2.0]$
Discrete Range:	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5, 1.75, 2.0]$

We expect a suppression value with a slow decay to lead to more newly created signature modules being selected and thus transmitted. Also, we expect that this will not have a large impact on the amount of data transmitted.

5.12.5 Static Moving Window Size Parameter (P3)

This parameter defines recency in the system. It is effectively a duration of time, to the multiple of 500 milliseconds over which a history of state data is stored. Initially we set each block size to 10 seconds; however, 500 milliseconds empirically showed a more noticeable effect on our test sizes and test durations. The default value is set at 60 or 30 seconds and ranges between 1 and 100, or 0.5 seconds and 50 seconds.

P3	
Default value:	60
Continuous Range:	$[1 - 100]$
Discrete Range:	$[1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20, 40, 60, 80, 100]$

The three use cases of the moving window are described in 5.9 and affect other key conditions. Shortening the accumulated danger window size reduces the likelihood of the under attack threshold T_{UA} being exceeded, depending of course on the input data. Thus, affecting the $P3$ parameter will affect that threshold. Reducing the two priority heuristic windows will equally reduce the duration over which the response and priority sums are accumulated. The $P3$ parameter does not affect the relativity of the two sum comparison, as the summed values are independent and compared to each other. Thus the effect is again on the time component of the comparison. Little to no intuitively useful information comes from varying the response and priority durations independently. Like the danger window decision, the time component of the heuristic decision is dependent upon the input data. Thus we can state that the same factors of experiment scale and duration and the order of input sources cause the $P3$ parameter to be a dependent variable, that depends only those external factors of its use case and the one internal factor affecting the T_{UA} threshold.

We can expect a larger $P3$ will cause the T_{UA} threshold to be exceeded more often, therefore the $P0$ volume percentage will be used more often and thus more data will be sent. However, we can expect a larger $P3$ parameter will equally increase the accumulated values of the heuristic decision, thus should show no noticeable independent effect upon the relative priority of the modules. Different data sources or orders of input can however, be expected to have different effects upon the accumulated danger classifications and upon module priorities and response rates.

5.12.6 Dendritic Cell Lifespan Parameter (P4)

This parameter defines the number of interactions a dendritic cell has before it is removed. The $P4$ parameter defines the lifespan and thus the impact of a new input on the permanent labelling of non-normal signature modules, as described in 5.8. The default value and range are arbitrarily set at 50 and between 1 and 100.

P4	
Default value:	50
Continuous Range:	$[1 - 100]$
Discrete Range:	$[1, 5, 10, 20, 50, 40, 60, 80, 100]$

We can expect a larger $P4$ value to lead to a faster rate of validating non-normal inputs and creating modules to distribute; the reverse is the case for a smaller $P4$ value. This has a direct effect on the sensitivity of the secondary classifier; although in this thesis, the classification evaluation is not of our direct interest.

5.13 Chapter Conclusions

This chapter stated the design of the CARDINAL-Vanilla architecture as an adaptation of Kim et al.'s CARDINAL abstract model architecture.

The description emphasises the engineered multi-agent system approach to the mathematical instantiation and the implementation design of the immunity-inspired model. These three distinct perspectives result in the definition of a complex system model of interactions and behaviours. The architecture co-operates across a network of devices, thus increasing the complexity of the complex system model.

The chapter defined the multi-agent system by its agent classes, interaction behaviours, environments and constraints. The interaction behaviours and process decisions are defined by the parameters and expression in logical set notation of individual cells as reactive agents.

The differences between the CARDINAL and CARDINAL-Vanilla models are summarised in section 5.2 and explicitly contrasted per item throughout the chapter. CARDINAL-Vanilla is the result of the change in problem domain, the result of mathematical instantiation and subsequently implementation, the essential completion of the algorithm and model specification and our interpretation of missing aspects of that specification. The model differences presented in this chapter are in process, process structure and flow control; in establishment of model parameters, variables, their values and ranges; in establishment of mathematical model equations (i.e. instantiation of the earlier model); in establishment of new decisions on ill-defined aspects of the earlier specification; and in establishment of the highly cohesive network transmission decisions.

The most valuable contribution of this work is the explicit definition of a novel probability-based heuristic for transmission of high priority information in decentralised peer-to-peer network applications.

From a high level perspective, the chapter has stated the procedures from input classification to response mappings, the signature module priority heuristic and network transmissions between the nodes of the distributed architecture that each contribute to the self-hardening system that is CARDINAL-Vanlla. The experimentation chapters that follow will evaluate the architecture for its immunisation capability on computer networks.

5.13.1 Next Steps

Next we will specify the evaluation framework and validation methodology used to evaluate distributed self-healing security systems and specifically CARDINAL-Vanilla. Following this we evaluate the architecture under real-world network conditions.

P.	Default	Range	Parameter Mapping Description: Model	Computation
			Antigen and Signals	
A_n			Antigen	Input data source attributes (e.g. [a="1",b="2",c="x",d="y"])
S_D	1		Danger signal	Attack label (e.g. label="attack1", label="attack2")
S_S	0		Safe signal	Normal label (e.g. label="normal")
S_U			Unknown signal	Unlabelled, i.e. unknown test data, (e.g. label="-")
			Maturation, Differentiation and Activation	
C_{COSTIM}	1		DC costimulation signal release to naïve T-cell in lymph node.	Increase TCN differentiation value toward CTL
C_{IL4}	1		DC interleukin 4 signal release to naïve T-cell in lymph node.	Increase TCN differentiation value toward Th2
C_{IL12}	1		DC interleukin 12 signal release to naïve T-cell in lymph node.	Increase TCN differentiation value toward Th1
T_{CTLMat}	1		Concentration required for CTL maturation	Threshold for TCN to CTL maturation
T_{Th2Mat}	1		Concentration required for Th2 maturation	Threshold for TCN to Th2 maturation
T_{Th1Mat}	2		Concentration required for Th1 maturation	Threshold for TCN to Th1 maturation
$P1$	2	[0.5-4]	Initial number of T-cell clones upon activation	Initial detector priority multiplier
V_{CTL}	$[P1 * T_{CTLMat}]$		Initial number of CTL cell clones differentiated from naïve T-cell	Initialised Type I danger detector priority value
V_{Th2}	$[P1 * T_{Th2Mat}]$		Initial number of Th2 cell clones differentiated from naïve T-cell	Initialised Type II danger detector priority value
V_{Th1}	$[P1 * T_{Th1Mat}]$		Initial number of Th1 cell clones differentiated from naïve T-cell	Initialised unknown detector priority value
			Proliferate and Suppress Detector Priority	
R_P	1		T-cell proliferation growth rate	Detector priority increase rate
$P2$	0.95	[0.1-4]	T-cell suppression	Detector priority reduction
R_R	$R_P + (RV_{Th1}/RV_{CTL})$		T-cell helper regulation of CTL	Detector priority regulatory increase
$P3$	60	$[1-100]$	Temporal T-cell commonness	Recent priority window size * 500ms
	"	"	Temporal T-cell antigen commonness	Recent response window size * 500ms
RV_*			Quantity of T-cell clones	Runtime priority value for T-cell agent
S_{RS}	Eq.5.4		Historical infection growth rate for T-cell	Sum of recent responses by TCE
S_{RV}	Eq.5.5		Historical clone growth rate for T-cell	Sum of recent priority value of TCE
			Cell Decay Rate	
$P4$	50	$[0-100]$	DC lifespan	Maximum number of interactions before removal
R_d	1		DC decay rate	Decrement value upon each interaction
			Sickness Focus for Volume Selection	
T_{UA}	50		Sickness definition	Under attack threshold
S_v	Eq.5.3		Current temporal sickness level	Sum of recent danger value
$P3$	60	$[1-100]$	Temporal sickness concentration, from danger inputs	Recent danger window size * 500ms
$P0$	0.75	$[0.25-1]$	Lymphatic flow concentration quantity during sickness	% detectors selected for transmission (A)
	"	"	Lymphatic flow distance during sickness	% destination nodes selected for transmission (A)
R_{NUA}	0.25		Lymphatic flow concentration quantity during no sickness	% detectors selected for transmission (B)
	"		Lymphatic flow distance during no sickness	% destination nodes selected for transmission (B)
			Probability Selection	
RNG_1			Chance of cell transport	Random number generator for selection.
S_1				Seed for RNG_1.

Table 5.2 – Table of CARDINAL-Vanilla parameters. Cell types parameters are described in text. Cells types are abbreviated as DC, TCN, CTL, Th2, Th1 in the table. Parameters are abbreviated as follows: A_* and S_* refer to data inputs, C_* refers to cytokine signalling, T_* refers to a threshold, V_* refers to an initial value, R_* refers to a rate, RV_* refers to a runtime variable and $P*$ refers to the key CARDINAL parameters under investigation.

Item	CARDINAL	CARDINAL-Vanilla
	Model Components	
Antigen role	"Attack signature"	Input without label.
Danger symptoms or attack signature	"Excessive CPU load, bandwidth saturation on network, abnormal email communication, etc"	Labelled inputs
Danger severity	Assessed by DCs	Determined by classifier.
Danger certainty	Assessed by DCs	Determined by classifier.
Threat level	"Derived from danger signal"	Derived from signal: damaging, safe or unknown
DC purpose	"Assess danger signals","Extract antigen" "Ascertain severity and certainty of attack"	Assess signals. Responder. Temporary container for antigen and signal.
Naïve T-cell	"Copy of antigen"	Temporary container for antigen and signal
CTL purpose	"Strong response", "Severe [w/] high certainty"	Respond to damaging input. Module with input signature mapping to response.
Th2 purpose	"Strong response" "Severe [w/] lower certainty"	As CTL, with different response mapping. Unused in experimentation.
Th1 purpose	"Weak response","Less severe " Assist proliferation of CTLs.	No response. Unlabelled input, will proliferate matching CTLs
T-Cell differentiation	"After a suitable period of time" "Exceeds a given threshold"	After the first maturation threshold is exceeded
Increase or decrease clone rate decision	"Compare historical infection growth rate against effector T-cell clone growth rate"	Same
Historical infection growth rate	None	Sum of danger inputs in a moving time window
Historical effector T-cell growth rate	"Total number of responses which peer hosts made in previous two time steps"	Quantity of matching responses in a moving time window
Unmatched received peer T-cells	"Suppress and create new naïve T-cell with lower maturation thresholds"	Suppress T-cell. Migrate to Tissue
Effector T-cell migration to Tissue	"Effector T-cells with positive clone values [migrate]"	All effector T-cell migrate
Effector T-cell migration to Peers	"Send effector T-cells with positive clone values and earlier received T-cells"	Select effector T-cells probabilistic– ally based on number of clones
Choose destination hosts	"Select randomly from all [.] peer hosts"	Same
Choose number of hosts	"Number of hosts selected [.] is determined by [.] the severity and certainty of attack."	If Historical infection growth rate is > threshold, then send more.
	Other Self-Healing Components	
Validation of input to response mapping	"After a suitable period of time"	Sum of danger value exceeds threshold, increased by repeated inputs.
Validation of network received detectors	After a number of matching inputs received, then received detector is validated.	Not specified
Generation of response	Not specified	Not specified
Sandbox validation of response	Not specified	Not specified
Issuance of response	Effector T-cell responds to matched antigen	Same

Table 5.3 – Table of key differences between the CARDINAL abstract model (Kim et al. 2005) and our CARDINAL-Vanilla model.

Chapter 6

How to Evaluate and Validate Distributed Self-Healing Security Systems

6.1 Introduction

This chapter contains principles of evaluating Distributed Self-Healing Security Systems (DSHSS) and the key issues necessary to evaluate the distributed CARDINAL-Vanilla architecture. The objective of this evaluation and validation methodology is to define the verifiable experimental design to measure the global performance of self-healing, under conditions that are feasibly similar to an Industrial Control System (ICS) corporate network scenario.

The CARDINAL-Vanilla architecture and our aim of a DSHSS require a distributed system evaluation methodology and testbed. At time of writing, we were aware of no experiment methodology for testing DSHSS applied to distributed security monitoring.

The sections of this chapter describe the experimental measures and metrics (6.2), measurement noise (6.2.5), experiment phases (6.3), individual node execution procedure (6.4), experimental constants (6.5), the datasets and input sources (6.7) and the heterogeneous user behavioural model (6.6) to reflect computer user behaviour. The resulting measurements are combined using multi-objective evaluation equations and methodology (6.10) to produce three system performance measures (6.9). Each experiment is conducted and evaluated in real-time (as described in 6.8). Later chapters will define the specific network node topologies, network sizes and the number of iterations used per trial.

6.2 Measurements and Metrics

From a distributed self-healing security system architecture, we want malware detectors and repair mechanisms to be propagated to all machines of our network as quickly as possible, while minimising impact upon other network traffic. These experiments are foremost to evaluate the efficiency and resource viability of CARDINAL-Vanilla in this scenario. We are interested in metrics representing an network-wide state or level of security, such that we only take account of global measurements

(as in "a concrete measurement extracted") and global metrics (as in "a calculation using extracted measurements") of data consolidated from all nodes of the architecture.

The three metrics that we are concerned with are:

(1) ***Distributed-M1***: quantity of monitored modules or detectors that are distributed to all network architecture nodes.
Range from 0 to $|ArbSig|$. Maximum values preferred. (**detector100Coverage*)
ArbSig is the set of inputs used to constrain the dataset, see 6.2.2.

(2) ***Time-M2***: time taken (in seconds) from module creation to full network distribution.
Range from 0 to ∞. Minimum values preferred. (**totalNetInocRateMedianSecs*)

(3) ***DataSent-M3***: data in megabytes (MiB) sent over the network by the architecture software (accumulated from each node) during each experiment trial.
Range from 0 to ∞. Minimum values preferred. (**dataSentTotalMiB*).

* Column name in result sets and logs.

6.2.1 Measures to Metrics

Several details that each of the metrics embrace are important to note. Each connected node has its own database of known detectors with various state timestamps and event logs (e.g. of sent and received network transmissions with their sizes in bytes). By the end of a trial run, a set of directories are created containing the experiment databases and logs for each node involved in the run. To extract the global metrics we combine these logs in a sensible and unbiased manner.

6.2.2 Detector Distribution Quantity Measurement (M1)

This metric measures the quantitative notion of units of useful modules produced by and distributed to all nodes of the networked security architectures. The notion of module quality is not measured by this metric. The metric is concerned with measuring usable modules only. In the *CARDINAL-Vanilla* architecture novel inputs become modules or detectors, in the manner stated in 5.8. Module quality is ignored by this measure as *CARDINAL-Vanilla* uses exact match detection, other systems may choose to change this. These inputs exist in a number of other states in a number of other locations; as in Table 6.1:

In metric M1 we are interested only when detectors can recognise and respond to new inputs. Such that M1 measures the number of detectors ($\|D\|$), where each detector $d \in D$ reached state (e). We show detector d in state (e) by d_e. Therefore expressed as:

$$M1 = \|D_e\| \tag{6.1}$$

In each node's logging databases we can collect the quantity of state (e) detectors and give each node a rating. However to make this a global measure, we filter out only those detectors found in all

State	Logged	Description	Model Location
(*a*)	*	read-in at *Sensor* from dataset	n/a
(*b*)	*	input received at Monitor	*Tissue*
(*c*)		input received at Analytical Engine	*Lymphnode*
(*d*)	*	detector created at Analytical Engine	*Lymphnode*
(*e*)	*	detector sent to Monitor, i.e. can respond	*Tissue*
(*f*)	*	detector selected for sending via network at Analytical Engine	*Lymphnode*
(*g*)	*	detector received via network at Analytical Engine	*Lymphnode*
(*h*)	*	detector responded to (classified) new input at Monitor	*Tissue*

Table 6.1 – Table lists the event states of inputs to detectors in the architecture. Abstract model locations are in *italics*. Events with an asterix (*) are logged within local log measurement databases, once per state per unique input. (c) is a transfer between agent environments and is not logged.

node logging databases for the trial run. Such that the unique quantity of matching detectors found at all nodes is the value of M1.

To constrain the search space (and therefore quantity) further, we preselected 20 arbitrary inputs, to form a set known as $ArbSig$, from the dataset and distinguished these as damaging and important. The purpose for this further arbitrary constraint is three fold; to evaluate the system against a range of rarely occurring (yet damaging) detectors, to represent real-world threats from this network-based security dataset and to permit human verification of correctness between repeated tests.

6.2.3 Detector Distribution Time Measurement (M2)

Following on from the quantity M1 metric input-to-detector states, the time metric (M2) uses these state timestamps to measure the interval duration between state (a) and state (e). M1 gave us a filtered set of detectors that reached state (e), which also are found on all nodes. From this we collect the latest arrival timestamp t_n for each of those detectors d_i. The earliest read-in timestamp – i.e. the earliest state (a) timestamp – for each of those detectors is selected as t_1.

The time interval ($Interval^{d_i}$) between t_1, the earliest state (a), and t_n, the latest state (e), for each detector $d_i \in D$ is then simply calculated, as:

$$Interval^{d_i} = t_n^{d_i} - t_1^{d_i} \tag{6.2}$$

Then, the median time interval duration of all detectors (D) is selected as our M2 value for this single trial run, as:

$$M2 = median(Interval^D) \text{ , where } Interval^{d_i} \in Interval^D \tag{6.3}$$

There is an event in which an input reaches state (e) but is not distributed to all nodes, i.e. not in all node logs contain a record of that detector in state (e). We choose to avoid the inclusion of a zero value, because this would suggest an implausibly perfect result. Instead a failsafe default is assigned,

this is the total experiment duration time for that node; often an order of magnitude larger. In this event, the M2 duration remains as the median from this set.

6.2.4 Detector Distribution Data Sent Measurement (M3)

To assess the feasibility in terms of impact on network traffic of each of the architecture designs the number of bytes sent from each node is measured and summed to produce a global (total) number of bytes sent during a trial run. The experiment procedure and constants dictate a requirement of 20 objects (6.5), of varying sizes, are sent by each node. Each object is measured and logged at the sending node and the receiving node.

The total sent bytes (B_{Sent}) recorded per object (O_j) at the sending node, per node (N_i) are accumulated, then converted to megabytes to give the value for M3:

$$M3 = \sum_{i=1}^{N} (\sum_{j=1}^{O=20} (B_{Sent}/1024^2)) \tag{6.4}$$

6.2.5 Sources of Noise affecting Measurements

Below is an incomplete list of expected causes for noise affecting collected measurements. Highlighted, in no particular order, are the more probable causes of noise and an estimate of their likely effect upon measured results.

Other Processes & Execution

Both virtualised network and real network testing are affected by behaviour of the operating system and other applications. Such as behaviour of thread execution, unexpected or scheduled application updates, time-dependent scheduled tasks and the maintenance tasks of other running processes. These items are feasibly uncontrollable and therefore can consider them random noise. Further testing, among other types, could look at the effects of other operating systems or more specifically, the underlying modules for handling execution.

Packet Loss

Similarly we can expect variation between trial runs caused by packet loss. Particularly on the loopback tests where CPU usage will be at a limit due to the number of concurrent CPU time intensive processes (virtual nodes) competing with the kernel to manage loopback messaging. According to (Alhomoud *et al.* , 2011), we can expect a roughly increasing number of dropped IP packets as the throughput amount increases. There will also be a saturation point for the real network tests, but this will depend upon the bottleneck cause and its threshold.

Transmission Failures

Other types of transmission failures are common in network tests, e.g. stream interrupts, connection disconnects, destination addresses unavailable, etc. These have been handled in the conventional network programming manner to attempt retransmissions after failures. The retransmission quantity is given in 6.3. These failures are likely to be spurious – low probability of occurrence – and uniformly random. They may affect the real network tests; however, are unlikely to affect the virtual tests as the Linux loopback device behaves with memory redirection which is not commonly prone to failure. The effect would manifest as delayed time to transmit – affecting time metric M2 – and in increased data sent but not received, which is specifically data metric M3.

Other Traffic

The enterprise network tests are run on operational workstations while connected to other switched segments of the enterprise. As such we cannot rule out the possibility of external access to workstations and switch facilitating the test and thus another form of noise out of our control.

Time

Time drift and time measurement precision is a potential source for small deviations in time measurements. This affects the real network tests, where many workstations may report differing timestamps at the same actual time. To minimise the drift effect and improve precision, a network time protocol (NTP) update is issued before tests begin.

6.3 Experiment Phases

The architecture is tested under a supervised learning experiment process, in which each node will execute four phases: *start-up*, *training*, *testing* and *shutdown*.

The *start-up phase*: each node is initialised and waits for an agreed period, this gives each node an opportunity to start each of its threads and have them perform their initialisation processes. The specific thread execution at each node is shown later in 6.4.1. The time duration is given in 6.5.

The *training phase*: begins the read-in of training inputs during which the labels (final column of dataset instance) is used to classify.

The *testing phase*: differs in that its inputs do not use (mask) the label. During training and testing each node transmits information to other nodes. The input limits and transmission quantity applied in these phases are defined in 6.5.

The *shutdown phase*: is a period dedicated to waiting other nodes to conclude their individual trial runs, such as allowing nodes to transmit their remaining queued packets to the neighbouring nodes. This period provides experiment robustness against the use of heterogeneity – heterogeneous device capacities and heterogeneous resource usages. The phase is concluded through an agreed message exchange between all connected nodes. The shutdown negotiation is described in 6.4.

6.4 Experiment Procedure

The experiment procedure that we execute to evaluate our self-healing architecture's immunisation rate is *Procedure J* shown in Figure 6.1.

The procedure steps through the four experiment phases (6.3) on each networked workstation or virtual machine. At the beginning of the procedure the following items are initialised: the input user model (I), the network transmission listener (R), the termination network listener ($wSTOP$) and the self-healing architecture processing algorithm (P) is started in the *start-up* waiting state. After P's waiting time is over P enters the *training phase* and begins parsing data inputs received from I.

In line with other constants defined in 6.5, P will process a number of inputs then schedule a Send Event (S). A Send Event is a network transmission of a set of modules to a set of nodes. Once the *training* and *testing phases* have completed on a given node, an experiment termination message ($sSTOP$) is sent to $wSTOP$ on all of its neighbour nodes. When all expected termination messages have been received, the node will terminate its own R and P ending its role in the experiment run. Typically, all nodes will terminate approximately simultaneously.

Next is a discussion of the development steps taken to reach this experiment procedure.

6.4.2 Experiment Procedure Development

Procedure J is the most robust of procedure designs that we achieved while investigating noise variations between runs. By robust we mean with respect to lowest standard deviation of measured results and number of transmissions sent and received by the nodes and in terms of reliably giving the shortest time duration per network experiment run in the virtual network tests. Due to the limited number of physical processors executing code the single machine virtual network tests were prone to resource blocking, particularly on IO waiting. The multi-workstation enterprise network tests did not experience such severe resource restrictions such that when using *Procedure J* on those machines we experienced fewer failures and better robustness (see chapter 7).

The benefits of *Procedure J* pay the cost of compromising on our design principles, Table 6.2 summarises this. A particular compromise was on the real-time principle of the study. Instead we moved toward a post-processing procedure (see 6.8). Others included refining experiment termination conditions, implementing multi-threaded and distributed system design patterns, upgrading hardware, delaying IO experiment logging, managing operating system restrictions and refining thread synchronization.

Changes such as *Procedure H* using short-life threads to process received inputs caused implementation level lock contention problems and impeded progress. Similarly, *Procedure A* using theoretical time delays to determine when send events were due to occur in practical fact, actually caused unreliable variation of delay duration much longer than the theoretical delays. The time delay issue was exacerbated as the number of current nodes increased in the virtual network tests. In standard enterprise networked testing the use of time delays as means to recreate a real-time testing scenario remain undependable in our experience, albeit to a lesser extent than in single machine tests.

Another of the hindrances in procedure design came as the number of architecture clients increased. The CARDINAL architecture's default connectivity is peer-to-peer and transmissions are status-dependent peer-to-peer. As messages are received and queued or processed the potential of

6.4.1 Experiment Procedure J

```
Version J:
        "POST_PROCESSING". 30000 inputs 20 fixed-input-interval transmissions. Wait for all received
        STOP messages before terminating.

Description of thread/ experiment behaviour:

    |-------------------------------------------------------------------------| t
    |-----------------------------------------------------------------| R / Pr
    |---| I
        | tI
    |*****|-----------------------------------------------------------| P
            |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | S
                                                                     | sSTOP
    |-----------------------------------------------------------------|....| wSTOP
                                                                          |-----| tR, tP.

    t           = Total time taken.
    R           = Listen/Receive network connections from other machines.
    Pr          = Process received network communication inputs.
    I           = Read-in 30000 lines (from dataset).
    P           = Process 30000 inputs (from I).
    S           = Make 20 network sending events. Send task issued after every 1500 inputs.
                   (20 unique serialised objects. Sent to 0..* machines.)
    sSTOP       = Send STOP message to all known hosts.
    wSTOP       = Await STOP message from all known hosts. On receive (all) shutdown R.
    t*          = Terminate each component of the experiment.
    --|..|      = End-point for this procedure is volatile.
    |**|        = Waiting time before beginning P.
```

Figure 6.1 – Diagram of thread behaviour and experiment procedure

self-imposed distributed denial of service (DDoS) becomes an issue that must be mitigated. During procedure development the DDoS effects requiring mitigation were upon memory taken by queued tasks and upon computational processing of received messages. *Procedure J* avoids these effects by increasing the physical memory resource, using smaller dataset sizes and limiting experiment tasks (e.g. transmission quantity) and duration.

v.	**Summary**
J	Set ulimit on open files and user processes, clear kernel cache between runs, turn swap off, use 16gb of RAM.
I	Use same thread for dependent tasks, use Actor and Immutable design patterns to issue queued tasks and execute tasks.
H	No logging to file during runs (avoid IO waiting time), cache opened sockets and streams, using short-life threads.
G	Shutdown terminations with countdowns and counts of sSTOP and counts of wSTOP.
F	Exit after X seconds of no receipt of data from input sensor. Shutdown conditions with 15% extra time.
E	Protocol based termination: send stop messages and acknowledgements. Remove ACK, Count wSTOP.
D	Send at fixed intervals. Terminate after 50% extra of time.
C	Use post-processing (no time delays).
B	Use 15% extra time before termination
A	Real-time simulation, with delay between input batches and send transmission every 3 seconds.

Table 6.2 – Summary table showing the experiment procedure version development of the distributed system test procedure.

6.5 Experiment Constants

The following are descriptions of the methodology constant values with justifications that led to those value decisions. Table 6.3 summarises the constants values. These selected values limit the duration and scale of tests. For example, architecture load, stress and life-long testing is not assessed. Other systems may choose to run a series of experiment trials, each with a different set of values.

P.	**Constant Name**	**Value**	**Unit**
t_{INIT}	Initialisation period	5	Seconds
I_{TRAIN}	Number of training inputs	15000	Instances
I_{TEST}	Number of testing inputs	15000	Instances
T_{TRANS}	Number of transmissions	20	Objects
$T_{Retries}$	Number of retries of a failed transmission	10	Retries
T_{Intv}	Input interval between transmissions	$\frac{(I_{TRAIN}+I_{TEST})}{T_{TRANS}}$	Instances

Table 6.3 – Summary table showing the experiment constants at each node of the distributed system under test.

The experiment initialisation period exists to provide enough time to ensure that all process instances have initialised, including having created their event logs and directories within which to store experiment measurement logs.

The quantity of experiment network transmissions is fixed to ensure tests are limited and repeatable. Twenty object transmissions, containing any amount of data content, was an arbitrary choice. Earlier trials identified that a fixed input interval is more robust (lower standard deviation over results) than a time-delayed interval.

The number of dataset inputs received at a node is limited to 30000 to enable the virtual network tests to execute, as limited by our experimental hardware set-up, and then to support comparisons between virtual and real network tests.

6.6 Distributed System User Behavioural Model

In real-world computer networks, including ICS and SCADA networks, it is frequently the case that individual workstations will create different behaviour profiles or patterns. Our objective was to recreate a scenario in our tests that would be representative. To achieve this, we implemented the simplest of user behavioural models to distinguish learning behaviour (such as experiences of malice inputs) of one user from another.

6.6.1 Three Parameter User Model

Our three parameter model was required to modify the inputs presented to each node of the distributed system while embracing our experiment design based upon a two-class classification data domain source. We wanted to incorporate user (node) profiled behaviour such that each user's behaviour could be defined prior to runtime and be repeatable for scientific testing. Our second aim is to embody a concept of susceptibility to attack.

The analogues of malware attack upon networked workstations and the antigen infection upon the human immune system intersect in the manner that damage is caused where its likelihood for causing damage is highest. A hacking attack (i.e. application probe fingerprinting and vulnerability analysis) and malware infection will target those areas with the highest probability of success. Similarly, in cellular theory, a damaging antigen's epitope molecules will interact with (effect) its corresponding site (an epithelial skin cell, for example). If the antigen meets a non-corresponding surface or a corresponding lymphocyte paratope then its damaging effect is less likely. This correlated intersection could therefore suggest that the susceptibility to attack is location, and therefore probability, dependent.

We can then go further, by drawing on our viewpoint published in (Scully *et al.* , 2013) that states that CARDINAL's (and CARDINAL-E's) entire architecture can be considered as a tissue surface in multiple regions on a network. For example, the gastrointestinal tract has the densest variety and quantities of immune system cells as a consequence of this region experiencing the highest quantities (probabilities) of infectious antigen (Murphy *et al.* , 2012). We might therefore expect that some regions are more likely to harden against infection (learn) due to more experience and thus this link of heterogeneous susceptibility into the user model becomes even stronger.

6.6.2 Static & Runtime Model Definition

The user model is defined by a static model ($\mathbb{M}_{\mathbb{STAT}}$) and a runtime model ($\mathbb{M}_{\mathbb{RUN}}$) with the parameters shown in Table 6.4. Of the three static parameters, the first parameter a is a percentage that represents the ratio of danger training inputs (to safe inputs), called the *anomalous proportion*. The second parameter s is a seed to a random number generator that will select input indices of instances in the dataset. The third parameter n_i is the node itself, distinguished by its index i.

Model	$P.$	Value(s)	Unit	Description
$\mathbb{M}_{\mathbb{STAT}}$	a_n	[0-1]	Float	Anomalous proportion for $n > 1$.
$\mathbb{M}_{\mathbb{STAT}}$	a_{n_1}	0.7	Float	Anomalous proportion for $n = 1$.
$\mathbb{M}_{\mathbb{STAT}}$	s	n_i	Int	Random seed to rng_1, rng_2.
$\mathbb{M}_{\mathbb{STAT}}$	n_i	$[0 - n`]$	Int	The unique user number per node, of $n` = n - 1$ nodes.
$\mathbb{M}_{\mathbb{RUN}}$	T	[0-∞]	Int	Time component.
$\mathbb{M}_{\mathbb{RUN}}$	P	[*tr*,*te*]	Nominal	Experiment phase (*training* or *testing*).
$\mathbb{M}_{\mathbb{RUN}}$	$a_{t_{tr}}$	[0-1]	Float	Percentage of danger inputs received by user during tr.
$\mathbb{M}_{\mathbb{RUN}}$	$a_{t_{te}}$	0.5	Float	Percentage of danger inputs received by user during te.
$\mathbb{M}_{\mathbb{RUN}}$	l_t	$[I]$	Set	Time dependent set of (I) instances from data source.
$\mathbb{M}_{\mathbb{RUN}}$	$\|\|d\|\|$	[0-∞]	Int	Data source instance quantity.
$\mathbb{M}_{\mathbb{RUN}}$	b	100	Int	Instance batch quantity.
$\mathbb{M}_{\mathbb{RUN}}$	v	0.25	Float	Maximum distance of noise variation upon $a_{t_{tr}}$.

Table 6.4 – Summary table showing the user model parameters. Values with square brackets show ranges, those without show constant values.

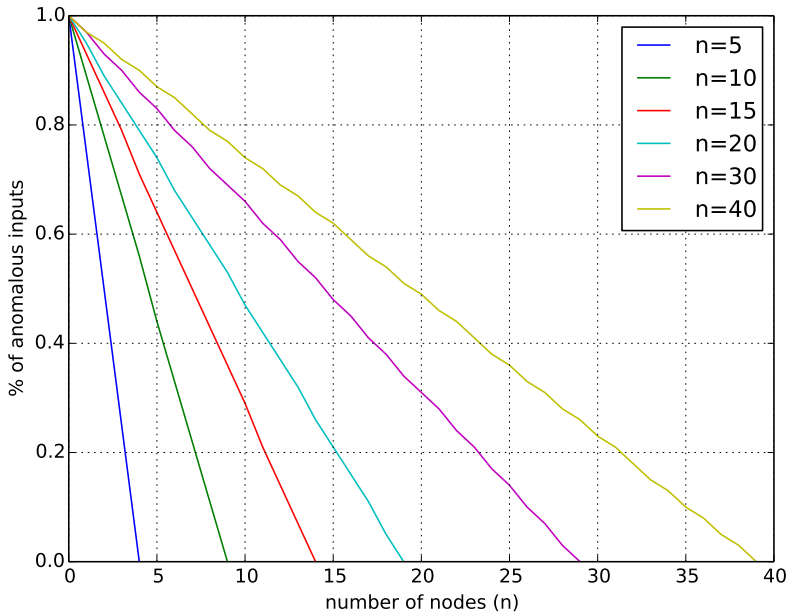


Figure 6.2 – Result of function $a(i, n)$, showing monotonic approximate linearity and close to even distribution of percentage values per user (node).

$$\mathbb{M}_{\mathbb{STAT}} = a(i, n) \quad , \quad s(i) \tag{6.5}$$

$$a(i, n) = \begin{cases} \dfrac{(n-1) - i}{n - 1} & , \quad \text{if } n > 1 \\ 0.7 & , \quad \text{if } n = 1 \end{cases} \tag{6.6}$$

$$s(i) = i \tag{6.7}$$

$\mathbb{M}_{\mathbb{STAT}}$ defines the static model as two functions. Equation 6.6 is the function to produce the a parameter of the user model. Figure 6.2 shows that a's value is approximately linearly distributed across all nodes of network sizes $n = 5, 10, 15, 20, 30, 40$. For n=1, we arbitrarily set $a = 0.7$. Equation 6.7

shows that s is selected by the node index of a given test.

$\mathbb{M}_{\mathbb{RUN}}$ defines the runtime model as four runtime variables produced using the static model $\mathbb{M}_{\mathbb{STAT}}$ parameters, T the time component and P the runtime experiment phase, i.e. the point in time during the experiment (either *training* or *testing*), as inputs into two functions. Both functions make use of random number generators (RNG) which are discussed later in this subsection. The first function $a_t(..)$ produces a percentage. The second function $l_t(..)$ produces a set of dataset instances from the dataset.

$$\mathbb{M}_{\mathbb{RUN}} = a_t \ , \ l_t \ , \ t \in T \ , \ P \tag{6.8}$$

$$a_t = \begin{cases} a_t(a, v, t, rng_1(s)) & , \quad \text{if } P = training \\ 0.5 & , \quad \text{if } P = testing \end{cases} \tag{6.9}$$

Equation 6.9 results in the value of a_t, which is the time-dependent runtime variable for parameter a. The value of a_t is also dependent on the experiment phase (P). During the testing phase a_t defaults to 0.5. Whereas during the training phase, a_t is given by the undefined function $a_t(..)$ which adds noise to parameter a during runtime.

The $a_t(..)$ function's output is a value that is $<= \pm 25\%$ the value of a. $v = 0.25$ is the maximum distance of variation. a_t is dependent upon the time step $t \in T$, where $T = \mathbb{N}_1 = [1, 2, 3, ..]$ (the natural set of integers). $rng_1(s)$ is the first random number generator taking the s parameter as seed input and gives a value between $a * 1.25$ and $a * 0.75$, in a deterministic manner as dependent upon t. a_t is thus not dependent upon earlier or later sequenced a_t values. Figure 6.3 shows the effect of this equation in a trial example of five users.

$$l_t = l_t(a_t, b, \|d\|, t, rng_2(s)) \tag{6.10}$$

Equation 6.10 is an undefined function that selects batches of line numbers from dataset sources. $b = 100$ is the batch length that we have selected for trials. $\|d\|$ is the dataset (file-specific) quantity of instances. $rng_2(s)$ is the second random number generator. The function selects $a_t * 100$ values between 0 and $\|d\|$ using $rng_2(s)$ in a time step t dependent manner. Figure 6.4 shows the effects of this function during a trial example of five nodes.

6.6.3 Distinguishing User Learning Experiences from User Behaviour

In section 6.3, we discussed the four experiment phases that occurred at each node under test, where phase (2) was training and phase (3) was testing. The user model defines the training experiences and behavioural experiences of each node.

Upon the training phase, the user model had three effects. (a) Different percentages (quantities) of danger (learning) inputs were read-in; by contrast, the remaining batch quantity of normal inputs were read-in. This effect was directed by model parameter a and a_t (see equations 6.9 and 6.6). The other effects were (b) different inputs were read-in and (c) different orders of inputs were read-in. Here (b) and (c) were caused by model parameter s as an input into the random number generators (discussed below) and produced by Equation 6.10.

During the testing phase, the (a) training effect was modified, whereas (b) and (c) are unchanged.

The (a) ratio of dangerous to safe testing inputs is set equally at 50%. This change to 50% allows for testing the learnt data and learning experience of the training phase in an unbiased and flexible way. For example, the effect of an alternative $a_t(..)$ function modelling a different user behaviour profile could be tested to measure classification accuracy or quantities of input responses per node. Whereas, the benchmark tests in chapter 7 test the collaborative learnt state after a sequence of 30 transmissions. Many variations to the training phase possible to test with this testing phase model.

6.6.4 Model's Use of Random Number Generators

The user model implementation makes use of two (of CARDINAL's three) instances of a linear congruential generator (LCG) algorithm to produce random numbers. Below we'll discuss how the two generators effect dataset input selection. Each are seeded by the model s parameter. As LCGs have well known pitfall cases, we analyse those pitfalls and discuss how they may effect our usage case in section C.2.

The first random number generator (RNG) is used to select dataset line numbers from each dataset file for each batch of b (100) inputs. The batches of inputs technique is used to avoid resource (i.e. open file handles, processes waiting for IO, etc.) hogging by any one process. Random selection of the dataset line numbers allows each user in the network test to receive a different sequence and different selection of inputs. We do this to represent differing user behaviour. While a uniform random distribution may not be a perfect representation for (typically Gaussian distribution) user behaviour, it does provides us with adequately different inputs per user.

The second RNG adds further noise variation between nodes in that the input batch quantity is varied by up to $\pm 25\%$. This precedes the first RNG in a sequential (deterministic) manner. As we have discussed, batches of (100) dataset inputs are read-in at a time. The model parameter a gives a percentage (and thus amount) of which each batch contains danger inputs. Thus, this noise variation specifically modifies a by $<= \pm 25\%$ (less than or equal to plus or minus 25% of a's value) for each input batch. The change is always from the static a value, such that each batch quantity change is independent from its previous change. The effect of this noise upon the user model adds a uniform random variation to the roughly linear distribution of the model's a parameter percentage values that are shown in Figure 6.2. Such is the case that we see batch sizes "hovering" above and below the user model percentage over a test run as in Figure 6.3 for $n = 5$.

The parameter a values of 1.0 and 0.0 remain unchanged by the noise variation to ensure that same maximum and minimum learning experiences occur at those nodes (users) in each test. This guarantees that the user with $a = 0.0$ will always depend on receipt of network transmissions from other architecture clients to learn. The user with $a = 1.0$ will therefore receive the greatest quantity of the inputs from the dataset and thus likely to rely least on the receipt of network transmissions.

The final effects of the user model can be visualised. Figure 6.4 shows the specific dataset coverage (line number index selection) read-in (via the model user) at each node while $n = 5$ during a benchmark (chapter 7) trial run. What we can see is that the bottom node (n_5) receives no danger (learning experience) inputs (100% missed). Whereas the top node (n_1) receives 75.11% ($100 - 24.89 = 75.11$), despite having its model parameter a value equal to 1.0. The reason for this is of course the effect of first RNG (independently) randomly selecting the line numbers. We can also visually recognise that the distribution is roughly uniform. Neither *darker* nor *lighter* areas are at the lower or higher numbers

in our range suggesting that the typical LCG pitfall are not affecting our usage case (refer to section C.2 for the context and scientific analysis of this discussion point).

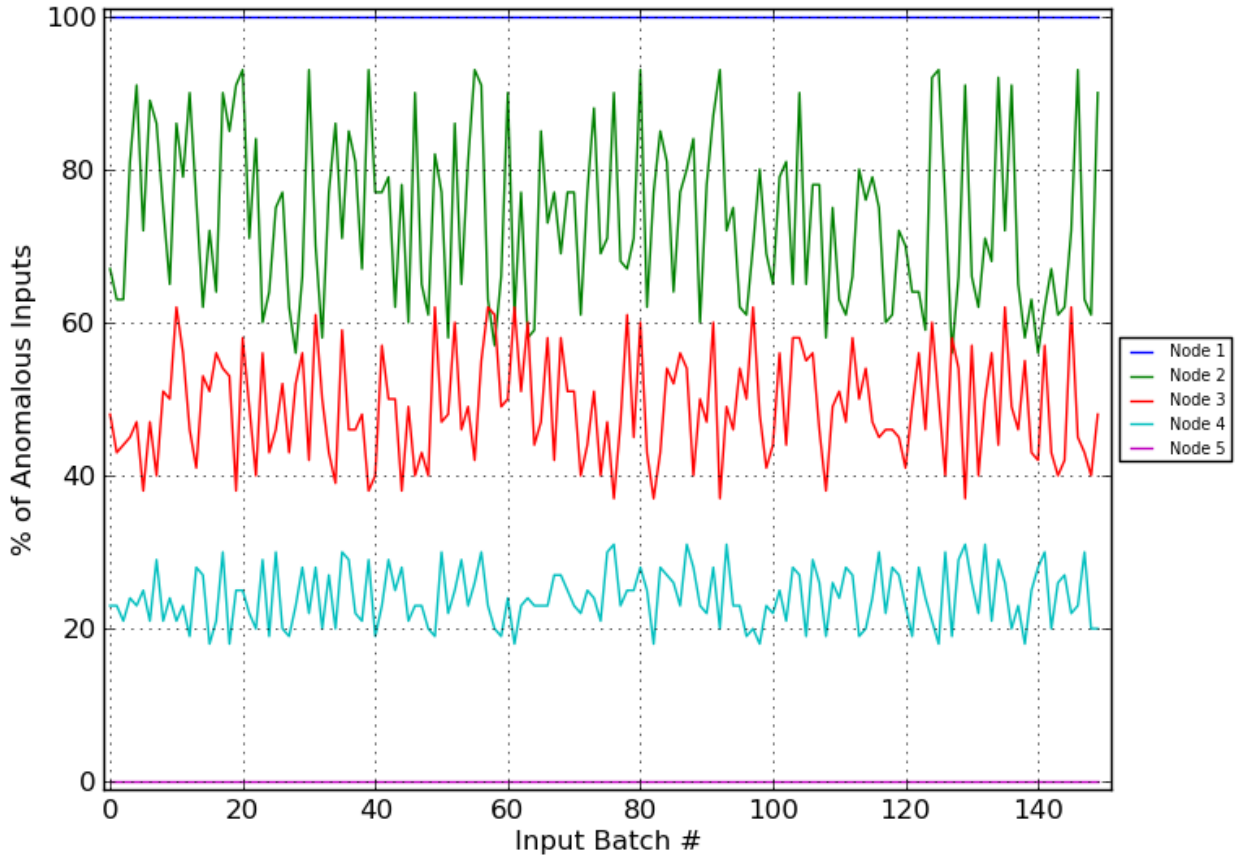


Figure 6.3 – Actual effects of noise variation upon the user model's learning experience percentage in a five node network test. Shown are percentages (y-axis) of anomalous inputs reported at each input batch request during a training phase (150 batches; 15000 inputs). Line colours represent behaviour at each node.

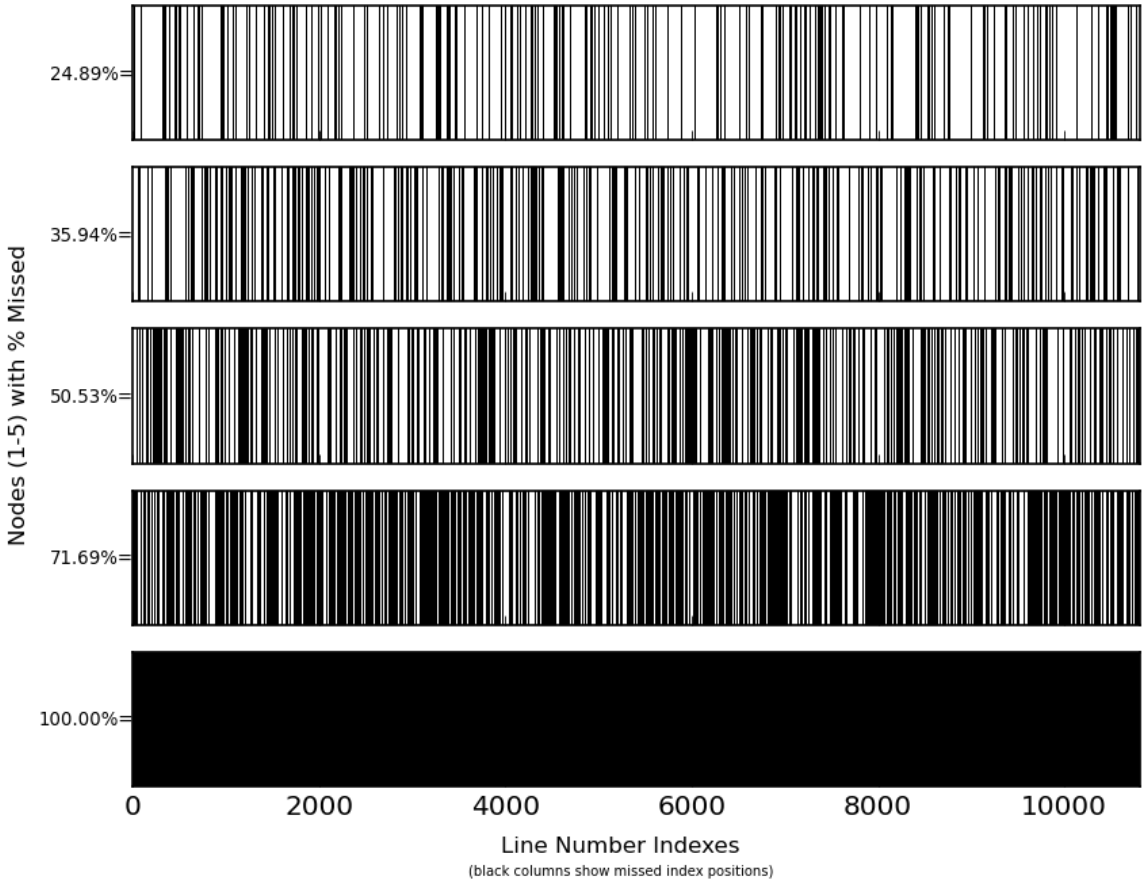


Figure 6.4 – Actual user model behaviour. White columns show the anomalous dataset line number indexes (instances) that were read-in, per node (horizontal strip). Black shows coverage of missed (not read-in) dataset instances.

6.7 Datasets

To test our hypotheses and architecture algorithm variants we needed a labelled data source or dataset. The thesis testing is focused on distribution and speed to full network immunisation, the data that is learnt is relatively unimportant until strong decision making is involved. However, ideally the data would be collected from several nodes on an industrial network whilst it underwent normal operational behaviour and also a variety of probing attacks and hacking. The ideal data content would be contextual data, network packet and stream meta and payload data, with transmission records, i.e. Siemens S7 network protocol stack data, of malicious binaries sent to the PLC(s) and the state data collected directly on the PLC bus(es), or less preferably collected via network transmission from the PLC(s).

At the time of experimentation we found no datasets containing network-based attacks directly on ICS networks or Supervisory Control and Data Acquisition (SCADA) networks. For completeness, the Mississippi State University 'ICS Attack Dataset' (Morris *et al.* , 2011) was publicly released in May 2013 and was created by transmitting commands separately to two ICS testbeds. It contains labelled measurements from a phasor unit (electrical waves), electrical relay states, simulated control panel states and Snort data logs. The dataset is known to be unsuitable for machine learning as "attacks were perform[ed] with gas pressure set to one value (x) and normal operation with another value (y) [SIC]", thus making its class separability unrealistic (Adhikari *et al.* , 2013)[1]. Secondly its attributes focused on electric sensor states of the automation system rather than detailed state data from the PLC's memory. The Snort log data items were very few, see our example and description in A.4. These attributes would not be indicative of most types of advanced persistent threat (APT) -like attacks nor their ongoing server communications, which are usually encrypted or obfuscated payloads on common protocols like HTTP.

We opted for a recent labelled dataset from the network security field. A number of substitute network security and vulnerability analysis datasets were available. The thesis evaluation tests report results using variants of the CSIC HTTP 2010 Dataset, as described in 6.7.1. Other datasets have undergone preliminary analysis in D.4.1.

6.7.1 CSIC HTTP 2010 Dataset Versions and Sampling

The dataset used in each of the thesis tests is the CSIC HTTP 2010 web penetration and hacking dataset consisting of normal and anomalous instances, created by Spanish Research National Council (Torrano-Giménez *et al.* , 2010b). The CSIC dataset is relatively recently created, labelled, applicable to both a collaborative network security defence problem and to network intrusion detection problems. Details of its creation can be found in their papers. Notably, the packets were generated from two databases of anomalous and normal inputs, as recorded during penetration testing upon a single domain address; this means order and frequency information is missing.

Prior to sampling and versioning for experimentation, the three file CSIC dataset of HTTP v1.1 packets was parsed into comma-separated value (CSV) format. This CSV format is used to input instances into the architecture nodes. The normalTrafficTraining and anomalousTrafficTest CSV files, of 104000 and 119585 instances, are selected. The normalTrafficTest file is discarded. The two selected

[1]The ICS Attack gas dataset can be modelled in < 1 seconds, see appendix D.4.1.

CSV files are split at their mid-point to produce two training and two testing files. The CSV files are sampled, described in the tables below, to form the datasets used in experimentation.

6.7.1.1 DATASET A ("Full")

The "full" dataset version converts all HTTP v1.1 protocol packet properties from the CSIC 2010 Http dataset in text format into CSV format. This leads to the following dataset properties:

The "full" CSIC dataset version:				
Qty of Attributes	17	17	17	17
Qty of Labels	1	1	1	1
Qty of Instances	59792	51999	59792	51999
Qty of Distinct Instances	59792	51999	59792	51999
Labels	anom	norm	anom_test	norm_test
Attributes	index, method, url, protocol, userAgent, pragma, cacheControl, accept, acceptEncoding, acceptCharset, acceptLanguage, host, connection, contentLength, contentType, cookie, payload			
Filename	sample_anom.csv	sample_norm.csv	test_anom.csv	test_norm.csv

Table 6.5 – The "full" CSIC dataset version converts each training and testing set into a corresponding CSV file.

Every instance in the "full" CSIC dataset is distinct as a result of the changing combination of *cookie* session ID, packet *index* number and its payload.

6.7.1.2 DATASET B ("Full v0.5")

The next dataset versions take the full dataset and first splits the *payload* (key=value) column into *key* and *value* columns. The number of columns is reduced to the three attributes and a label. The "full v0.5" dataset version takes all instances from "full" CSV formatted dataset with the reduced attribute quantity and leads to a dataset with the following table of properties:

The "full v0.5" CSIC dataset version:				
Qty of Attributes	3	3	3	3
Qty of Labels	1	1	1	1
Qty of Instances	59792	52000	59792	52000
Qty of Distinct Instances	2142	30	693	30
Labels	anom	norm	anom_test	norm_test
Attributes	url, payload_key, payload_value			
Filename	anom_sample.csv	norm_sample.csv	anom_test.csv	norm_test.csv

Table 6.6 – The "full v0.5" CSIC dataset version converts each training and testing set into a corresponding CSV file.

Fewer instances in the "full v0.5" CSIC dataset are distinct as a result of reducing the number of attribute columns. The normal training and testing CSIC files have fewer distinct combinations among these attributes and directly reflect CSIC's published dataset source files. The anomalous test file has fewer distinct instances than the anomalous training file. We reason that the cause of this is due to the

distribution of distinctness across the original anomalousTrafficTest file; the first-half has more distinct instances, the latter-half has fewer.

6.7.1.3 DATASET C ("v0.5")

The "v0.5" dataset takes a uniform random samplings of the original normalTrafficTraining and anomalousTrafficTest sources to produce these files. The anomalous training set and testing sets use an arbitrary 9% sampling value. The normal training set uses 9% and the normal testing set uses another arbitrary 35% sampling value.

The "v0.5" CSIC dataset version:				
Qty of Attributes	3	3	3	3
Qty of Labels	1	1	1	1
Qty of Instances	10812	9065	10812	36137
Qty of Distinct Instances	615	30	615	30
Labels	anom	norm	anom_test	norm_test
Attributes:	url, payload_key, payload_value			
Filename	anom_sample.csv	norm_sample.csv	anom_test.csv	norm_test.csv

Table 6.7 – The "v0.5" CSIC dataset version takes a uniform random sub-sampling of the "full" dataset and reduces the quantity of attributes to three descriptive HTTP protocol data properties.

Evidently the sampling produces a matching quantity of distinct instances in both anomalous training and testing sets and normal training and testing sets. We reason that this coincidence is due to uniform selection of the more common instances. This is expected as there are many 'singularly' (1 occurrence) distinct instances and fewer 'common' (> 1 occurrences) distinct instances. Further, this must be a result of the original source dataset creation process, that in turn used random and arbitrary sampling.

6.7.1.4 DATASET D ("v0.5.2")

The "v0.5.2" dataset's files are drawn from the original normalTrafficTraining and anomalousTrafficTest files. The first 9% is the anomalous training set and the second 9% (9-18%) is the testing set. The first 9% is the normal training set and the second 10% is the normal test set. Both 9% and 10% are arbitrarily selected.

The "v0.5.2" CSIC dataset version:				
Qty of Attributes	3	3	3	3
Qty of Labels	1	1	1	1
Qty of Instances	10812	9065	10812	10034
Qty of Distinct Instances	615	30	615	30
Labels	anom	norm	anom_test	norm_test
Attributes:	url, payload_key, payload_value			
Filename	anom_sample.csv	norm_sample.csv	anom_test.csv	norm_test.csv

Table 6.8 – The "v0.5.2" CSIC dataset version takes an ordered sub-sampling of the "full" dataset and reduces the quantity of attributes to three descriptive HTTP protocol data properties.

6.8 Real-time vs Post Processing

The experiments in the later chapters follow a conventional post-processing procedure. A post-processing procedure is generally understood to mean performing an action on a set of data at a time after the data has been collected. We follow that definition and in addition we intend it to mean parsing sets of data (from a pre-collected dataset) that have already been loaded on to heap space memory. The post-processing approach is simple and relatively less time consuming to validate than a real-time and real input system, i.e. analysing live network packet streaming data.

The post-processing procedure, i.e. labelled data, has directed the verification and repeatability of the experiment phases (6.3) presented in this chapter. Post-processing, i.e. inputs loaded on the 'heap', has guided the measurement collection procedure, experiment (execution) procedure (6.4) and user behavioural model (6.6) implementation. Whereas the real-time and real input system has guided the measurement methodology (6.2).

6.9 Multi-Objective Evaluation Metrics

We can summarise the M1 to M3 metrics to distinctly show one algorithm's comparative performance over another's. Our goal is to find an optimal configuration of parameters and algorithms that maximises our system performance measure; however, we need to carefully select the combinations and weightings of each set of metric data such that they satisfy our research objectives.

Three metric combinations were chosen to evaluate the self-healing system's immunisation rate. Evaluation of the immunisation rate is made without consideration of data transmissions (O1). Then for evaluation of the immunisation rate on a low-throughput capacity network (O2), such as a SCADA network. Finally the system's immunisation rate is evaluated on high-throughput capacity networks (O3).

The first combination (O1) is agnostic to transmission throughput, thus it is assigned a weight of 0. The latter (O3) is more tolerant (lower weighting) to the data transmission quantity than the O2 objective. The selection of 0.1 as the M3 weight value in the O3 objective is arbitrary. This weight is based on a belief that the algorithms can be applied to higher throughput enterprise computer networks that provide an increased throughput capacity at an order of 10 over the low throughput networks.

The three system performance measures are combined as shown in Table 6.9, each with their weightings. Each metric's preferred semantic order of scores is explicitly given in Table 6.10. The semantic ordering of each measure is essential to take account of in order to combine the scores, as we shall discuss in the next section.

Immunisation Rate Objective		**Weightings** $M1$	$M2$	$M3$
For generalised immunisation	(O1)	1	1	0
For low throughput networks	(O2)	1	1	1
For high throughput networks	(O3)	1	1	0.1

Table 6.9 – Table of real value weightings per metric for each of the 3 objectives.

Metric		Preferred Semantic Condition
Distributed-M1	$(M1)$	MAX
Time-M2	$(M2)$	MIN
DataSent-M3	$(M3)$	MIN

Table 6.10 – Table of the semantic preferences per metric. MAX indicates a larger value is preferred over a smaller value. MIN indicates the reverse.

6.10 Combined System Performance Measures

Having established three measure combinations in section 6.9, we shall next investigate how to combine those measures, leading us to a comparable system performance evaluation value.

In this section we explain and extend Rodriguez and Weisbin's bit-based system performance evaluation approach (Rodriguez & Weisbin, 2003) with two extensions. Both extensions add weighting and add handling of the semantic measure condition. A reliable equation extension (see 6.10.3) that removes the bit-based system (key to Rodriguez's approach) is presented.

Table 6.11 summarises the core of our two extensions and Rodriquez's original equations. The variables and functions of the Ratio of Distances approach are fully described in the following text. In brief the $p(k,1)$ refers to the benchmark score and $p(k,m)$ refers to the actual score for a given measure k leading us to a ratio score. The ratio scores per measure are then combined to give our immunisation rate scores. Section F.1.1 critiques the Log Ratios with Inverses approach and it weakness to provide reliable results while combining MIN semantic condition metrics.

Semantic Condition	**Ratio of Distances** *(see 6.10.3)*	**Log Ratios with Inverses** *(see F.1.1)*	Rodriguez & Weisbin
MAX	$\left(\frac{p(k,1)-p(k,m)}{p(k,1)}\right) \times w_k$	$\log_2\left[\frac{p(k,m)}{p(k,1)}\right] * w_k$	$\log_2\left[\frac{p(k,m)}{p(k,1)}\right]$
MIN	$-\left(\frac{p(k,1)-p(k,m)}{p(k,1)}\right) \times w_k$	$\log_2\left[1/\frac{p(k,m)}{p(k,1)}\right] * w_k$	-

Table 6.11 – Summary table of the evaluation equations used to combine multiple metric scores into a combined system performance measure score.

6.10.1 Approach

Rodriquez and Weisbin's (Rodriguez & Weisbin, 2003) present an approach to combining multiple performance measures using an accumulation of log base-2 values of measurement ratios, using a control value as ratio denominator. This logarithmic ratio shows performance difference of *primitives* (i.e. metrics) between a *reference system* (a control) and several test systems (i.e. other algorithms). The logarithm brings the measures into the binary number base; as an outcome the control algorithm's bit value score becomes 0. Using the logarithm leads to small evaluation scores for result sets similar to the benchmark control result set. Scores above zero indicate better performance than control and worse performance are awarded scores below zero.

6.10.2 Rodriguez & Weisbin's Equations

We will follow Rodriguez's conventions during this discussion of their equations and our extensions. k refers to a given metric of the N set of metrics and m is a given system of M systems, where M_1 is our reference system.

Rodrigiuez & Weisbin's log base-2 ratio equation is given by,

$$P^{'}(k,m) = \log_2 \left[\frac{p(k,m)}{p(k,1)} \right] \tag{6.11}$$

where $P^{'}(k,m)$ is the *bit* value score given by the base-2 logarithm upon the ratio of the $k-th$ measure (metric) for the $m-th$ system (algorithm) over the first system, where 1 is the reference system. Rodriguez's final bit score $s(m)$ function accumulates each $P^{'}(k,m)$, then halves the results; reproduced and adapted for clarity as (6.12):

$$s(m) = \frac{1}{2} \sum_{k=1}^{N} P^{'}(k,m) \tag{6.12}$$

where N is equal to the number of measures and $P^{'}(k,m)$ is given by Equation 6.11.

At this stage we can evaluate each system with equal priority upon each measure, where larger values are semantically better than smaller values (MAX) for all measures. However, in our measures of time taken and data sent, we prefer smaller values over larger values (MIN). In our objective O3, we weight some measures more heavily than others. Both cases need to be added for our purpose.

In section 6.10.3 we show a safe solution to achieve combining performance measures.

6.10.3 Ratio of Distances

The $P^{''}(k,m)$ function (Equation 6.13) is added taking the negation of a ratio of the distance from the reference value (where the reference value is the ratio denominator) to handle this MIN semantic measure condition case, as:

$$P^{''}(k,m) = -\left(\frac{p(k,1) - p(k,m)}{p(k,1)} \right) \tag{6.13}$$

An additional weight argument w_k is added to handle measure priority in (6.14). The weight is specific to a given k measure.

$$P^{''}(k,m,w_k) = -\left(\frac{p(k,1) - p(k,m)}{p(k,1)} \right) \times w_k \tag{6.14}$$

To ensure the ratio scores for MAX are directly proportional to MIN the $P^{'''}(k,m,w_k)$ function (Equation 6.15) is added taking the ratio of the distance from the reference value:

$$P^{'''}(k,m,w_k) = \left(\frac{p(k,1) - p(k,m)}{p(k,1)} \right) \times w_k \tag{6.15}$$

A conditional function $P(k, m, w_k, c_k)$ is added to handle both measure semantic types:

$$P(k, m, w_k, c_k) = \begin{cases} P''(k, m, w_k), & \text{if } c_k = MAX \\ P'''(k, m, w_k), & \text{if } c_k = MIN \end{cases} \tag{6.16}$$

Bits No Longer

You will notice we have removed the logarithm. Standard ratios, as Rodriguez's approach capitalises upon, will give a *pivot-point* of 1 (where $0 < 1$ implies worse performance and > 1 implies greater performance). Where as distance ratios give a pivot-point of 0, leading to negative values for performance worse than the reference.

In this case we have a few options to reincorporate the logarithm. First, taking the logarithm of a negative number will give a complex (Cartesian coordinate) number result. The complex number's components (*real* and *imaginary* values) can then be squared, summed and square-rooted to give a real number distance (from the point of origin). Figure 6.6 shows this mapping back to real numbers for base-2 logarithms of $[-10, ..9]$. Relative interval distances between negative, zero and positive values become lost. An available option, that we choose not to attempt, is to interpolate the negative numbers into a known normalised range and subsequently deal with relative ratio interval distances of these (already highly manipulated) logarithm distance values.

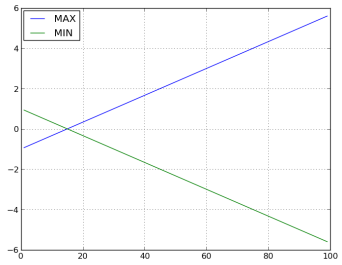


Figure 6.5 – Distance ratio output from $P(k, m, w_k, c_k)$ for MAX and MIN semantics for range $[1, ..100]$ from $r = 15$.

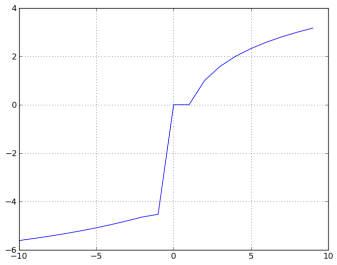


Figure 6.6 – Real number mappings of base-2 logarithms of $[-10, ..9]$.

A second option to reintegrate the base-2 logarithm is to use a form of linear interpolant on the results of $P(k, m, w_k, c_k)$, Equation 6.16. The function gives a linear output (see Figure 6.5) such that each could be raised (by a constant C) into a new range and entered into the logarithm. In such a case, Rodriguez's score function ($s(m)$, Equation 6.12) could then be used (exchanging $P'(k, m)$) to combine and compare the single measure. Due to the arbitrarily selected constant used to enlarge the values, the option also remains untested.

Each of the approaches add complexity, unnecessarily so. Both can result in zero and sub-zero values which, when entered into a logarithm and subsequently translated from Cartesian space back to real number space, result in incomparable relative intervals above and below zero, as was shown in Figure 6.6. Alternatively an arbitrarily-sized constant is issued. This approach fails when the constant increase is exceeded by a large-enough ratio difference. Thus the ratio of distances approach is not a candidate for us as a binary number unit evaluation mechanism.

Combining without Bits

Instead of bits, we can now combine the outcomes of Equation 6.16 for every $\{k, m_i\}$ in an alternative way.

If we rank each k measure (according to its semantic condition as either MAX or MIN) with maximum numbers indicating better rank, we can then sum the total ranks per m system (with weights) to give a combined maximal (optimised) rank. In this approach interval distance information is lost. For example, if m_x has an improved distance of 500 shown in k=1 and k=2, k=3 have deficit distances of -1, whereas if m_y has k=1 distance of 1 and k=2, k=3 have distances of -1, then both m_x and m_y rank equally.

The solution we will pursue maintains interval distances and updates Equation 6.12 with our described conditional distance ratio function $P(k, m, w_k, c_k)$ (Equation 6.16), in place of Rodriguez's $P^{'}(k, m)$ function:

$$s(m) = \frac{1}{2} \sum_{k=1}^{N} P(k, m, w_k, c_k) \tag{6.17}$$

Here we accumulate the weighted distance ratio scores from Equation 6.16. By keeping the $\frac{1}{2}$ fraction we simply reduce the distance in the accumulated score space. Each k, m is weighted by w_k and its measure semantics addressed by c_k, as above.

Example Calculation of Objective 3 (O3)

We will take a short opportunity to show with a simple example how the newly developed equations perform with the data in Table 6.12. We start by describing the example systems and then entering each system column of performance measure values (e.g. $[10, 10, 10]$) with their respective weights and condition values into Equation 6.16.

The first columns M_2 and M_3 in Table 6.12 show equal measure distances from the reference system M_1. Overall, M_2 is improved over the reference system by distance 5 on k=3 and M_3 is (equally) worse than the reference system on k=3; k=1 and k=2 are equalised by both systems. Columns $M_4, ..M_7$ show scaling distances for the equally weighted measures k=1 and k=2 (equal measures for k=3) giving each of these four systems a combined total improved distance of 5.

Performance	Systems								
Measure (k)	M_1	M_2	M_3	M_4	M_5	M_6	M_7	Weighting	Condition
1	15	10	20	16	17	18	19	1	MAX
2	15	10	20	11	12	13	14	1	MIN
3	15	10	20	15	15	15	15	0.1	MIN

Table 6.12 – Table of example data for reference and test systems (M_i). The quantity of metrics and conditions of weights (w) and semantic measure conditions (c) match our third objective (O3). The individual performance measures (k) can be thought of as M1, M2 and M3 metrics. Systems (M_i) can be thought of as alternative algorithms, where M_1 is a benchmark algorithm.

Table 6.13 shows the output of this first function for each system measurement. In M_2 and M_3 for $k = 1$ and $k = 2$ we expect counteracting values for a given system (e.g. $[-0.333, 0.333]$). These cases have equal weighting, opposing semantics and equal values for the reference system (e.g. $[15, 15)$

and each test system (e.g. $[10, 10]$). We then find that the third measure output to be one-tenth that of the second measure output. The reference system evaluates to 0 for each of its measures, as it compares distance to itself and evaluates to 0. Systems M_4 to M_7 evaluations scale with equal intervals (a measure distance of 1 equates to 0.066 in ratio space) and monotonically in both directions.

Performance Measure (k)	Systems M_1	M_2	M_3	M_4	M_5	M_6	M_7	Weighting	Condition
1	0.0	-0.333	0.333	0.066	0.133	0.2	0.266	1	MAX
2	0.0	0.333	-0.333	0.266	0.2	0.133	0.066	1	MIN
3	0.0	0.033	-0.033	0.0	0.0	0.0	0.0	0.1	MIN

Table 6.13 – Table of results from our $P(k, m, w_k, c_k)$ performance measure function in Equation 6.16 for individual measure data from Table 6.12. Results are truncated at 3 decimal places.

We can now accumulate the final results of Equation 6.17 in Table 6.14, which sums the $P(k, m, w_k, c_k)$ function outputs and multiplies by .5 per system.

Performance Measure	Systems M_1	M_2	M_3	M_4	M_5	M_6	M_7
O3	0.0	0.033333	-0.033333	0.333333	0.333333	0.333333	0.333333

Table 6.14 – Table of results from $s(m)$ score function in Equation 6.17. Results are truncated at 6 decimal places.

As shown in Table 6.14, $M_4, ..M_7$ systems, with equal individual performance measure distances, are translated into equal accumulated score space distances. With some confidence, we can now use this approach to combine separate measures into a single system performance measure.

Summary

By extending Rodriguez's evaluation equation, we have shown the distance from a reference as a ratio over that reference, can be used to fairly combine measures with weightings into a single system performance measure. This has removed Rodriguez's binary unit evaluation mechanism due to the complexity it adds when dealing with negative logarithm inputs.

6.10.4 Conclusion for Combined System Performance Measures

The Ratio of Differences approach in section 6.10.3 has been shown to be reliable for comparing algorithm performance. This solution removes the binary unit evaluation component of (Rodriguez & Weisbin, 2003) paper's methodology.

6.11 Chapter Conclusion

This chapter has stated our designed evaluation and validation testing components used to evaluate the CARDINAL-Vanilla platform architecture and has described the required principles for testing future distributed self-healing security systems.

Each test will execute the experiment testing procedure in 6.4, stepping through each of the four phases in 6.3 while using the experiment design constants stated in 6.5. The distributed user behavioural model will give each architecture node a different set of dataset inputs, in the manner stated in 6.6. The cyber security testing datasets used in these thesis evaluations have been expressed with descriptive analysis in 6.7. The architecture experiments are measured by a set of global metrics described in 6.2.

To simply and intuitively evaluate the self-healing architecture's immunisation rate performance we have combined the metrics into objective system performance measures for different types of network conditions. This measure of system fitness enables optimisation of a single combined metric to quantify the improvements or costs of new features, as we have described in 6.9. The two sets of equation methodologies used to combine those global metrics have been analysed in 6.10, where only the Ratio of Differences is can be recommended for reliability in future case studies.

The limitations of the measurements at this stage consider the most pressing issue to immunise the system as rapidly as possible, thus our metrics quantify distribution and signature quantity as a measure of protection level. In Chapter 8 we introduce other metrics, including recovery rates. Classification metrics have not been included here due to the specific aims of the thesis and platform. Global measures of detection quality can be locally monitored and verified during runtime via a supervised learning method. These metric additions will be key when expanding self-healing component testing.

The limitations of the distributed user profile model are led by the single behavioural model. Adding alternative distribution or more accurate cyber security user models will in future enable more rigorous testing of applied self-healing system robustness.

6.11.1 Future Work upon the User Model

To aid the architecture-wide classification evaluation a more convention training and testing dataset split would be expected instead of the 50% split. The k-fold cross validation is a suitable candidate method to separate the source dataset into training and testing datasets (Mosteller & Tukey, 1968). The quantities of training inputs and testing inputs would therefore change to reflect the ratios given by the number of folds. For example, 10-fold cross validation would lead to 90% of all inputs entering during training and 10% during testing. The total input quantities, as shown in Table 6.3, will therefore be adjusted.

The effect of an alternative line number allocation function, $a_t(..)$, could be investigated. Idealistically, further work on modelling the user profile behaviour on the target network would be a far better solution than reading-in from a model, but that is time consuming and particularly challenging to label accurately.

6.11.2 Next Steps

Next we will describe the evaluations, beginning with a benchmark of the architecture and evaluation methodology, and combining a comparative analysis of the architecture.

Chapter 7

Self-Healing Benchmark

7.1 Introduction

In this chapter we will evaluate *CARDINAL*-Vanilla as a self-healing system and our validation methodology. In an effort to address our hypothesis set in section 1.5, the objective of this work is to evaluate the performance of the architecture's bio-inspired priority heuristic and selection mechanisms for distribution of security modules against a set of equivalent engineered approaches. Secondarily the experiment will provide a detailed use case description of the validation methodology and its experiment design.

The following is a restatement of the key hypothesis:

> "In the context of a distributed and self-healing system, compared to engineered approaches a reactive multi-agent and holistic immune system inspired approach will have better distribution, and thus capability for self-healing, performance over a range of networked environments."

The *CARDINAL-Vanilla* architecture is specifically under analysis. It provides the distributed self-healing system platform and the reactive multi-agent and holistic immune system inspired approach. In section 7.2 we evaluate CARDINAL-Vanilla against three other selection and distribution algorithms under a virtual network environment. This is followed up in section 7.4 with a comparison of the AIS and engineered algorithms on a real enterprise network. As the results between the two network type conditions show some discrepancy, we include further analysis of the two sets of results in section 7.6.

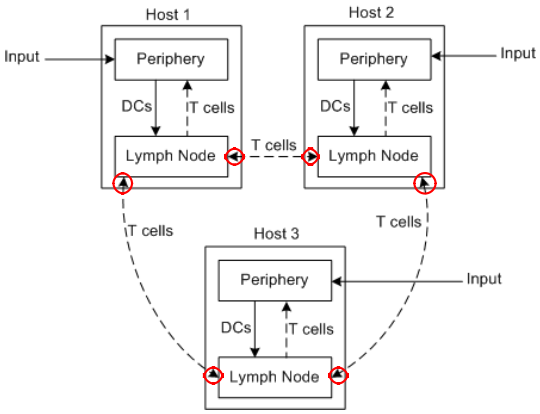


Figure 7.1 – Diagram of distribution (transmission) components under test circled in red, within the CARDINAL model.

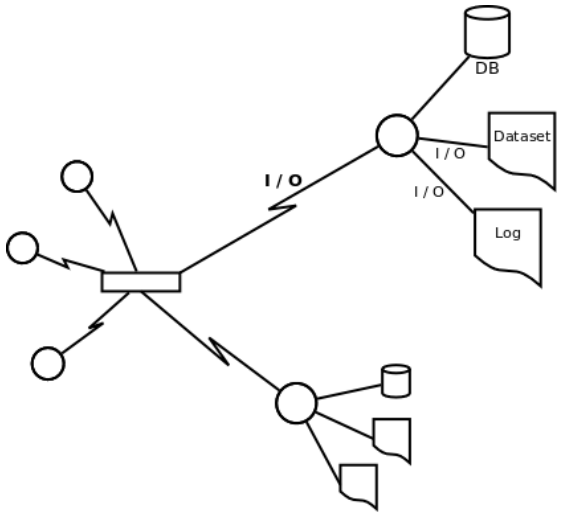


Figure 7.2 – Network topology example of five peers on a single network segment and experiment infrastructure for the self-healing system test. *DB* is a runtime database used to store the detectors, a *log* file (per instance) collects data from that instance which is later parsed to calculate test measurements.

7.2 Comparison on Virtual Networks

7.2.1 Objectives

In this work we will conduct the evaluations under a virtual network test condition. Under test is a comparison of the detector and response distribution technique given by *CARDINAL* in a default configuration against three other techniques. Figure 7.1 shows the highlighted distribution components and abstract multi-agent system locations within the model at which the comparative algorithms are operating. Figure 7.3 shows a flow diagram of the procedure to select detectors for each transmission as used by each networked client of the architecture.

The three other techniques include our engineered solution which will send only newly acquired detectors and two baseline algorithms. The first is not networked and the second is a *naïve* algorithm that will send all of its known detectors at every opportunity.

To make the comparison we shall use the following measurements, as formally stated in 6.2. The quantity of detectors distributed to all hosts on the network which we call *Distributed-M1* or simply *M1*. The time taken to ensure all nodes on the network have a given detector and its response, which we refer to as *Time-M2* or *M2*. Finally, the amount of data sent over the course of the test to achieve the immunised state, which is *DataSent-M3* or *M3*. We will combine these measures to give a self-healing system *immunisation rate*, as described in 6.9 using the equations stated in 6.10.3. These give an evaluation score of each approach to measure suitability for the three types of network conditions and applications.

7.2.2 Algorithm Comparisons

In these tests the algorithms we are assessing directly replace *CARDINAL*'s transmission of detectors process, as detailed in section 5.10. The four algorithms are as follows:

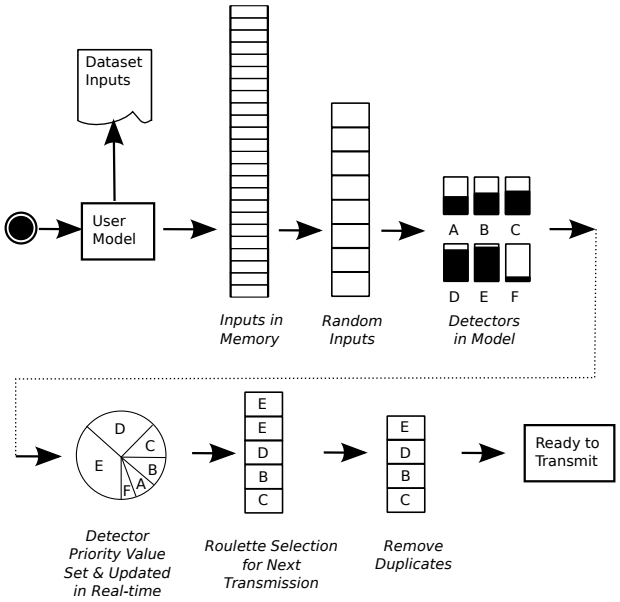


Figure 7.3 – Flow diagram of CARDINAL's detector selection process used during each transmission event.

Algorithm 1: CARDINAL-Vanilla

CARDINAL-Vanilla (Vanilla) is modelled upon the immune system using variable selection of quantities and probabilistic selection of detectors and destination hosts. A description of the distribution algorithms was given in 5.10 and is summarised by Figure 7.3. As a reminder, a roulette wheel approach is used to select detectors, where those with a high 'Clonal Rate' (number of copies) have higher probability of selection. Selection of destination hosts is uniformly random. Whereas the percentage quantities (of hosts and detectors) is calculated at run-time for each transmission, determined by the *under-attack* condition.

Algorithm 2: Optimal-Static

Optimal-Static is our approach engineered as optimal for this test scenario. This algorithm maintains a memory state of which detectors have been sent, such that they will never need to be resent. It uses no state knowledge of which detectors are on any other machines such that it matches the *localised knowledge* approach of CARDINAL and that of the immune system. This approach is so named as it would be sub-optimal under dynamic scenarios, for example if any transmissions or hosts fail or delete their detectors. Using the hash table representation, the increase in local time complexity created by the lookup of every detector (before it is sent) is O(n)={O(1)*O(n)} and in space complexity for storage of this algorithm's hash table is O(n) in every case, irrespective of the quantity of connected hosts; where n is equal to the number of local detectors.

Algorithm 3: NaïveSendAll

NaïveSendAll is the first of two baseline algorithms. NaïveSendAll is designed to always send all known data at every sending event (which, as we will describe below, are on 20 occasions throughout each test). This benchmark maximises the amount of data sent and shows the maximum number of detectors that can be sent during each given test configuration.

Algorithm 4: NoNet

NoNet is the second baseline algorithm, designed to never send any detectors and terminate when it has processed its allotted quantity of inputs. Foremost, its results show the impact of the heterogeneous user agents upon the detector quantity measurement, and giving evidence within the results that the experiment design supports heterogeneity and the original research question. As a side effect, it gives us the maximum time between the first input of the first signature and the last time a CARDINAL instance (client) terminated its experiment run.

7.2.3 Experiment Design

This experiment design specialises upon the more in depth design, experiment methodology and implementation details that we described in chapter 5. In these experiments the labelled Dataset D (specified in 6.7.1) is read into each architecture node client. The *CARDINAL* architecture which is run locally on a single host computer. n instances (clients) of the implementation are executed and connect using TCP ports (from 60175-*) to each of the other clients in a fully connected peer-to-peer topology (see subsubsection 7.2.3.4). Figure 7.2 gives a diagram of the network topology and experiment infrastructure.

Each client is configured with a standard configuration, including a fixed quantity and read-in rate (quantity per processed batch) of inputs from the dataset and a pre-built set of neighbouring client (internet or port) addresses. Per client, a unique behaviour model effects the *danger-to-safe ratio* of inputs from the labelled dataset. This gives each client a heterogeneous user experience and differing learning input experience from the dataset, as discussed in section 6.6.

The following sections will add focused detail upon this experiment setting including the implications of measurements, experiment runtime procedure, the dataset and network protocol and key configurations.

7.2.3.1 Implications of Measurements

Measurement data are logged locally to avoid impacting network traffic and to reduce the scaling effect of I/O disk logging upon the trial metric scores. After a trial has completed, the experiment logs are created, collated and parsed to extract global measurements. Further detail of the metrics is in section 6.2. An experiment's performance is evaluated according to objectives, described in detail in section 6.9 using the equations stated in 6.10.3.

We expect uncontrollable noise variation in terms of CPU activity, disk clean-ups, package updates, other network communications, etc. between tests running on live operating systems. These noise items are further described in 6.2.5. We have preferred medians over means as a centrality performance indicator to reduce the effect of this noise. Twenty-five trials per test were selected as an initial quantity to overcome the spread of variation in each algorithm and noise. Each run was carried out sequentially, taking anything from 6 to 50 minutes from the beginning of one run, to the beginning of the second, depending on the quantity of instances and the algorithm under test.

7.2.3.2 Experiment Runtime Procedure

The experiment follows procedure J, which is described in section 6.4.1 and by Figure 6.1.

This is a post-processing experiment, in the sense that the dataset is read into memory before any dataset instance is processed and in the sense that there is no delay between inputs, as one might find in typical computer network behaviour. This removes the I/O delay caused by file socket and stream queuing and access from the *Time-M2* time measure. After reading each 1500 inputs from memory, a network transmission request is issued, collected and carried out to send a set of T-cell detectors to other (virtual) network instances in the connected *CARDINAL* architecture. After sending 20 transmissions (sets of detectors) and having processed 30,000 inputs from the dataset (including training and test phases), a STOP protocol message is sent to all other instances on the (virtual) network. When all machines have received a STOP from all other machines, they each will terminate, thus terminating the experiment run.

The procedure follows a simultaneous and distributed training–testing experiment methodology. That is a training phase followed by a test phase, where each node performs this same procedure with differing subsets of data. Each node performs their individual learning procedure at approximately the same time and rate. In addition each node shares their learnt T-cell detectors assisting the network wide learning task. The measure of classification correctness (e.g. accuracy) is not of our current interest; however, measuring the rate of distribution is our evaluation goal and thus our experiment procedure design facilitates that goal.

7.2.3.3 Dataset

Dataset D is used these trials, statistical dataset details are available in 6.7.1. The CSIC 2010 HTTP dataset used in web application firewall (WAF) feature selection and classification experimentation in (Nguyen *et al.* , 2011), (Torrano-Gimenez *et al.* , 2011) and its initial design is described in (Torrano-Giménez *et al.* , 2010a). The conversion process to our Dataset D version (*v0.5.2*) results in the attributes {URL, Payload_Key, Payload_Value, Label} and is in section 6.7.1. We use this dataset as it comprises of recent (2010) attack vulnerabilities, is relevant to network security and has labelled instances. The attributes used here are simple to extract with Berkeley Packet Filter (BPF) parsing of packets (e.g. TCP, HTTP) network inputs on each node, such that the transition from dataset to network device packet filtering is straightforward.

7.2.3.4 Network Topology

The network topology was a single virtual network segment as Figure 7.2 exemplifies, consisting of n virtual machines (up to 20). Each test used a single computer with n instances of the Java virtual machine (JVM), we give specific version information below. Each JVM used socket communication over the local loopback device (IP address 127.0.0.1) and attached using the TCP protocol for application addressing (e.g. on ports 60175 to 60179) and message transmission.

7.2.3.5 Network Protocol

The theory behind the network communication of *CARDINAL* (and as such, our test implementation) is the transmission of activated T-cells. Therefore, upon receipt of a T-cell an action is taken to incorporate the T-cell into the local decision system. Two protocol message types are used taking advantage of the (Java 7) language's built-in serialisation and marshalling of an object stream. A

received object's class name is used to trigger an associated action to the given protocol message. The first message type is a data structure object containing a number of selected T-cells which, upon receipt, are incorporated into the local decision system as described in section 5.10. The second type is a STOP state marker. This informs another machine that the sender is in a ready state to terminate its test run. Terminating before other nodes are ready leads those other nodes into a potentially irrecoverable state, this second message type avoids this computational problem state.

7.2.4 Configuration

Below are the key configurations of these experimental trials. The virtual machine environment and execution script configurations are described in section D.3 and in section D.1.

7.2.4.1 Time Desynchronisation

A 450 millisecond delay is applied between each instance execution, this was our technique to preserve host number order. An effects analysis of this decision is undertaken in 7.6.

As shown by example in section E.4, execution of background processes in bash is non-deterministic. The order was required for us to track individual host behaviours (as defined in 6.6) and to report metric data per host over many trials. The time offset (between first and last instances) is calculated as 450ms*(n-1), such that the final host for 10,15 and 20 host test is delayed by {4.05, 6.3 and 8.55} seconds. By accounting for the constant initialisation period duration (6.5) and, as gauged from empirical experience, that network transmissions usually begin after the first 5 seconds at $n = 15$ or $n = 20$, we can make an educated assumption. Theoretically, the delay would have a negligible effect upon the transmission successes, leading to the time metric (*Time-M2*), in the virtual network experiments at $n = 15$ and $n = 20$, and zero-to-negligible effect at $n = 10$ with a 4.05 second offset.

7.2.4.2 CARDINAL Configuration

The immunological model parameter values that *CARDINAL-Vanilla* requires have been set in Table 5.2 and described in 5.12, such as the differentiation thresholds for CTL, Th2 and Th1, the percentage of detectors to distribute while not under-attack and many others.

7.2.4.3 Network Hosts Configuration

Each instance had a unique network configuration file and separate from the *CARDINAL-Vanilla* configuration. Firstly this contained a set of neighbouring port addresses on the loopback device. For example in the five node virtual network test, the first node had four neighbouring port addresses as 127.0.0.1:60175 to 127.0.0.1:60179.

This configuration file also contained the parameter values of the distributed user model specific for this node, as detailed in 6.6 and the experiment constants defined in 6.3. The noteworthy user-specific model parameters are the random seed $\mathbb{M}_{\mathbb{STAT}_s}$ – that feeds the user model and the roulette wheel selection mechanism with CARDINAL-Vanilla, and the anomalous proportion $\mathbb{M}_{\mathbb{STAT}_{a_n}}$ that drives learning.

7.2.4.4 Benchmark Parameter Defaults

We discussed each of the key parameters of *CARDINAL* in 5.12, labelling them parameter 0 to 4 (P0-P4). In this experiment series, the values for P0-P4 were set as {*P0=0.75, P1=2.0, P2=0.95, P3=60, P4=50*}. These values were chosen arbitrarily, after conducting a series of initial trials.

7.3 Results

We present the metric scores and immunisation rate performance scores of the four algorithms. Table 7.1 contains the raw metric results and Table 7.3 contains the combined immunisation rate scores. Medians (η) and standard deviations *(Std)* over 25 trials are presented over the five network sizes (n) of 1,5,10,15 and 20 virtual clients. The medians are used to show the independent middle trial from each algorithm's results. The standard deviations are used to show the range of noise impact on and/or variation over an algorithm's set of the trials.

In tables 7.2 and 7.4 we present the difference significance results between the two algorithms. We use Mann-Whitney U statistical tests to measure difference between the AIS algorithm (Vanilla) and the engineered algorithm (Optimal-Static). The Cohen's d effect size is presented to give a measure of difference scale. The trial run scores of the two algorithms, per network size are inputs into the equations. The inferential statistical test selection process and the complete description can be found in section E.1 and the Mann-Whitney U test in subsection E.1.1.

7.3.1 Metric Performance

The following three subsections will report the performance of the algorithms on the metrics *Dist-M1*, *Time-M2* and *DataSent-M3*.

7.3.1.1 Number of Detectors Distributed to Entire Network (M1)

In this experiment the *Dist-M1* score has a theoretical maximum of 20, as defined by the number of instances in (`arbitrary_signatures.csv`) the accompanying signatures dataset file. Within the 9% sample anomalous dataset 17 reside. However, 14 is our observed maximum quantity of matches that, we explain, is due to randomised line number selection used by each of the user agents. At network sizes 1,5,10,15 and 20 the observed maximum *Dist-M1* values are 11,13,14,14 and 14.

In the single instance tests (n=1), all algorithms perform equally on *Dist-M1*, the quantity of distributed detectors. The maximum values for *Dist-M1* on the remaining virtual network size tests are consistently from Optimal-Static and SendAll, showing best performance with [11,13,14,14,14] detectors distributed.

A clear rising and falling trajectory in Vanilla's *Dist-M1* results is apparent, as Figure 7.5a highlights. The bar plots in Figure 7.4 show this trajectory over all networked trials (5-20 nodes), where n=10 visually shows the lowest spread and the highest centrality performance for Vanilla. For n=15 and n=20 lower medians and greater standard deviations are clearly found. When compared to other algorithms, higher standard deviations on *Dist-M1* suggests less performance consistency from Vanilla's algorithm design or greater noise during its trial runs.

We also observe two inconsistencies in Optimal-Static's results. Optimal-Static's 17th trial at n=15 resulted in an *Dist-M1* value of 0. Optimal-Static's 21st trial at n=5 resulted in an *Dist-M1* value of 12. These two runs are Optimal-Static's only variation shown under the *Dist-M1* results.

In all virtual network sizes, SendAll's results show the lowest standard deviation and highest medians and very closely followed by Optimal-Static. In fact, if we exclude two of Optimal-Static's anomalous runs as outliers, we can conclude that Optimal-Static's standard deviation and median results are equal to that of SendAll at all network sizes.

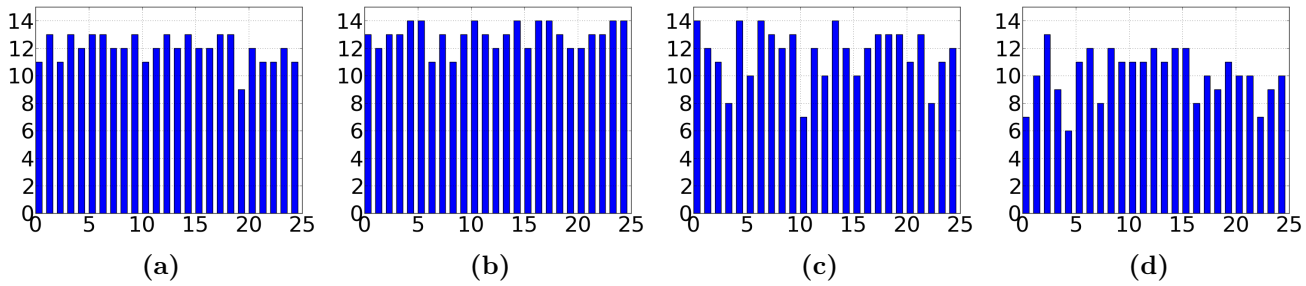


Figure 7.4 – *Dist-M1* variation of Vanilla performance over 25 runs with **(a)** 5, **(b)** 10, **(c)** 15 and **(d)** 20 nodes. Y-axis shows quantity of detectors (M1) from 0 to 14 (max), X-axis shows the test run number from 1 to 25.

The Mann-Whitney U test results for difference between Vanilla and Optimal-Static on *Dist-M1* data are in Table 7.2, showing virtual network sizes 5 to 20 with significant difference, where the critical probability value is 0.05 and very large (VL) effect sizes in favour of Optimal-Static over Vanilla. The performance difference at n=1 was insignificant with no effect in Cohen's d equation.

7.3.1.2 Time Taken to Distribute Median Detector (M2)

The *Time-M2* score is the measured time duration between a detector being first classified and the time at which it has been distributed to all other nodes on the network. Therefore values must be non-negative and smaller values are preferred, as discussed in section 6.2.

The minimum values for *Time-M2*, the median time to distribute the median (timed) detector to all network nodes, are SendAll for 5 node virtual network size and Optimal-Static for sizes 10, 15 and 20 nodes. Note that the NoNet *Time-M2* time results for sizes 5,10,15 and 20 are the duration of time taken to complete the experiment, such that we can exclude them from our analysis comparisons. That is, only in the event that zero detectors are found on at least one machine (which the experiment design creates if no distribution occurs), for all other known detectors (read-in elsewhere on the network) the median experiment total time value is used. From this set of time durations, the median time value is presented as the *Time-M2* value for the given trial. This is, by design, always the case for the final node under our malicious user-agent design, as that node reads-in 0% malicious inputs, consequently creating zero detectors. Therefore, the NoNet *Time-M2* time results are exactly the duration of time taken, by the median node, to complete the experiment.

Optimal-Static's 21st trial at n=5 was a cause of variation in the *Dist-M1* results; however, within the *Time-M2* results a variation is not distinctive. The 21st trial *Time-M2* result is 37.152, where the value range is from 17.763 to 38.529 seconds.

Optimal-Static's 17th trial at n=15 resulted in an *Time-M2* value of 214.371, the largest value in the trials. If we exclude this as an outlier the value range becomes 108.961 to 137.439, the standard deviation then becomes 6.012 which a reduction of 13.076 seconds (in deviation) or 31.5% of this test series' standard deviation value.

Vanilla's *Time-M2* median performance ranks at [2nd,2nd,3rd,4th,4th] on each network size test, when compared to the other algorithms. It's standard deviation ranks at [4th,2nd,4th,3rd,3rd]; however if we remove the excludable Optimal-Static outlier, that rank list becomes [4th,2nd,4th,4th,3rd].

A general observation in all algorithm results is that of a monotonic increase in median and standard deviation values, if we exclude Optimal-Static's outlier (17th trial at n=15), as the Figure E.1 box and

whisker plot shows. This is expected behaviour for this metric over these tests given our problem domain.

The Mann-Whitney U test results on time taken (*Time-M2*) data in Table 7.2 show significant difference between the AIS and engineered algorithms on sizes n=1,10,15,20. The 5 network node size results showed a small (S) effect size from Cohen's d equation in favour of Optimal-Static (by its smaller mean value), but with no significant difference ($p = 0.385$) from the Mann-Whitney test. At n=1, a small (S) effect is implicated in favour of Vanilla, with a significant P value. In virtual network sizes 10,15 and 20 the results show very large (VL) effect sizes and significant P values from each statistical tests, in favour of Optimal-Static.

7.3.1.3 Total Data Sent During Experiment to Distribute (M3)

The *DataSent-M3* score is the total amount of data sent in megabytes (MiB) by all nodes during an experiment. As with time taken, the values must be non-negative, smaller values are preferred and a full discussion can be found in section 6.2.

The minimum median values on each network size test were consistently given by Optimal-Static, if we exclude NoNet's zero communication results. Vanilla sends considerably less data than SendAll on all network sizes, however more than double that of Optimal-Static. This can clearly be seen in the median values of Figure E.2b. Optimal-Static sends [40%,37%,38%,39%,39%] of the data of Vanilla in each network size test.

Optimal-Static's 17th trial at n=15 resulted in an *DataSent-M3* value of 45.3, the smallest value in the trials. If we exclude this as an outlier the value range becomes 49.5 to 51.5, the standard deviation then becomes 0.506 which is 44.3% of the reported test series' standard deviation value.

Centrality and spread are monotonically increasing in all algorithm results with *DataSent-M3* data, including Optimal-Static when we ignore the outlier.

The Mann-Whitney U test results on data sent (*DataSent-M3*) data in Table 7.2 show significant difference on all network sizes in favour of the engineered algorithm over Vanilla. In virtual network sizes 1,5,10,15 and 20 the results show very large (VL) effect sizes and consistent significant P values (well below $p = 0.05$) from the statistical tests, in favour of Optimal-Static.

7.3.2 Immunisation Rate System Performance

Each of the immunisation rates system performance measures are described in section 6.9. In the following sections the three scoring rate results are presented. The ratio of differences equation (explained in subsection 6.10.3) is used to calculate each algorithm's performance using the median SendAll algorithm result as (reference) ratio denominator. Interval distances are comparable as ratios relative to the reference.

NoNet's results in these O1,O2,O3 evaluations are of little to no-interest as its quantity of distributed detectors (*Dist-M1*) are 0 for all network sizes above n=1; for comparisons we can therefore ignore its results.

n	**Vanilla**		**SendAll**		**NoNet**		**Optimal-Static**	
	η	*(Std)*	η	*(Std)*	η	*(Std)*	η	*(Std)*
				Dist-M1 - Quantity				
1	11.0	(0.0)	11.0	(0.0)	11.0	(0.0)	11.0	(0.0)
5	12.0	(1.0)	13.0	(0.0)	0.0	(0.0)	13.0	(0.2)
10	13.0	(0.935)	14.0	(0.0)	0.0	(0.0)	14.0	(0.0)
15	12.0	(1.952)	14.0	(0.0)	0.0	(0.0)	14.0	(2.8)
20	10.0	(1.824)	14.0	(0.0)	0.0	(0.0)	14.0	(0.0)
				Time-M2 - Time (Sec.)				
1	0.228	(0.467)	0.48	(0.171)	0.037	(0.038)	0.356	(0.224)
5	30.24	(3.733)	27.48	(4.25)	51.729	(1.954)	30.515	(4.213)
10	93.263	(9.429)	83.425	(6.833)	106.116	(3.717)	78.214	(5.052)
15	190.317	(17.103)	146.085	(10.418)	156.071	(4.037)	123.279	(19.088)
20	308.347	(24.994)	254.701	(27.708)	201.217	(5.487)	183.182	(9.207)
				DataSent-M3 - Data (MiB)				
1	0.5	(0.05)	3.7	(0.091)	0.0	(0.0)	0.2	(0.0)
5	11.7	(0.609)	63.1	(1.31)	0.0	(0.0)	4.3	(0.137)
10	55.8	(1.768)	306.8	(5.992)	0.0	(0.0)	21.4	(0.321)
15	128.5	(4.89)	742.4	(13.851)	0.0	(0.0)	50.4	(1.141)
20	242.1	(5.019)	1327.1	(36.324)	0.0	(0.0)	93.1	(0.958)

Table 7.1 – Tabular results of metrics M1,M2,M3 showing medians (η) and standard deviations (*Std*) (with n-1 denominator) over 25 iterations of the four algorithms.

n	U	df	P	**Significance**	**Cohen's d**	**Effect Size**
			Dist-M1			
1	Nan	24	Invalid	insig.	0	None
5	121.5	24	5.21011297493^{-06}	sig. < 0.05	-1.33128047094	VL ($M_X < M_Y$)
10	100.0	24	4.70860459449^{-07}	sig. < 0.05	-1.57383180959	VL ($M_X < M_Y$)
15	73.0	24	1.53826232707^{-07}	sig. < 0.05	-0.729230295609	VL ($M_X < M_Y$)
20	0.0	24	4.42074243841^{-11}	sig. < 0.05	-3.0394599246	VL ($M_X < M_Y$)
			Time-M2			
1	161.0	24	0.00169458420376	sig. < 0.05	-0.171464805294	S ($M_X < M_Y$)
5	297.0	24	0.38550931209	insig.	0.0351322388539	S ($M_X > M_Y$)
10	36.0	24	4.27340833535^{-08}	sig. < 0.05	2.05021940858	VL ($M_X > M_Y$)
15	24.0	24	1.14836069702^{-08}	sig. < 0.05	3.33476931569	VL ($M_X > M_Y$)
20	0.0	24	7.07828112425^{-10}	sig. < 0.05	6.71157900329	VL ($M_X > M_Y$)
			DataSent-M3			
1	0.0	24	2.00381441843^{-11}	sig. < 0.05	9.61665222414	VL ($M_X > M_Y$)
5	0.0	24	5.99175394206^{-10}	sig. < 0.05	16.74674193	VL ($M_X > M_Y$)
10	0.0	24	6.75903458446^{-10}	sig. < 0.05	27.226769181	VL ($M_X > M_Y$)
15	0.0	24	6.99564862571^{-10}	sig. < 0.05	22.2852090264	VL ($M_X > M_Y$)
20	0.0	24	6.88265973526^{-10}	sig. < 0.05	41.6269462567	VL ($M_X > M_Y$)

Table 7.2 – Mann-Whitney U test results for CARDINAL vs Optimal-Static of Metrics M1,M2,M3 over 25 iterations with Cohen d's difference and effect size.

n	**Vanilla**		**SendAll**		**NoNet**		**Optimal-Static**	
	η	*(Std)*	η	*(Std)*	η	*(Std)*	η	*(Std)*
				O1				
1	0.263	(0.476)	0.0	(0.175)	0.461	(0.039)	0.129	(0.228)
5	−0.095	(0.086)	0.0	(0.076)	−0.941	(0.035)	−0.055	(0.078)
10	−0.08	(0.081)	0.0	(0.04)	−0.636	(0.022)	0.031	(0.03)
15	−0.204	(0.118)	0.0	(0.035)	−0.534	(0.014)	0.078	(0.16)
20	−0.217	(0.106)	0.0	(0.053)	−0.395	(0.011)	0.14	(0.018)
				O2				
1	0.695	(0.475)	0.003	(0.181)	0.961	(0.039)	0.602	(0.228)
5	0.313	(0.086)	0.017	(0.077)	−0.441	(0.035)	0.411	(0.078)
10	0.33	(0.08)	0.002	(0.036)	−0.136	(0.022)	0.496	(0.03)
15	0.208	(0.116)	0.003	(0.032)	−0.034	(0.014)	0.544	(0.159)
20	0.193	(0.106)	−0.002	(0.048)	0.105	(0.011)	0.606	(0.018)
				O3				
1	0.306	(0.476)	0.0	(0.175)	0.511	(0.039)	0.176	(0.228)
5	−0.054	(0.086)	0.0	(0.076)	−0.891	(0.035)	−0.009	(0.078)
10	−0.039	(0.08)	−0.001	(0.04)	−0.586	(0.022)	0.078	(0.03)
15	−0.163	(0.118)	0.001	(0.035)	−0.484	(0.014)	0.125	(0.16)
20	−0.176	(0.106)	−0.001	(0.053)	−0.345	(0.011)	0.187	(0.018)

Table 7.3 – Tabular results of objectives O1,O2,O3 showing medians (η) and standard deviations (*Std*) (with n-1 denominator) over 25 iterations of the four algorithms.

n	U	df	P	**Significance**	**Cohen's d**	**Effect Size**
			O1			
1	161.0	24	0.00169458420376	sig. < 0.05	0.171464805294	S ($M_X > M_Y$)
5	240.0	24	0.0812060036749	insig.	−0.470465368049	M ($M_X < M_Y$)
10	26.0	24	1.43477475231^{-08}	sig. < 0.05	−2.09972060965	VL ($M_X < M_Y$)
15	25.0	24	1.28383893526^{-08}	sig. < 0.05	−1.88109304568	VL ($M_X < M_Y$)
20	0.0	24	7.07828112425^{-10}	sig. < 0.05	−5.00406611943	VL ($M_X < M_Y$)
			O2			
1	204.0	24	0.0180607040578	sig. < 0.05	0.0510519519454	S ($M_X > M_Y$)
5	117.0	24	7.73018370803^{-05}	sig. < 0.05	−1.17235307613	VL ($M_X < M_Y$)
10	1.0	24	7.98333592845^{-10}	sig. < 0.05	−3.0407796079	VL ($M_X < M_Y$)
15	25.0	24	1.28383893526^{-08}	sig. < 0.05	−2.26760857705	VL ($M_X < M_Y$)
20	0.0	24	7.07828112425^{-10}	sig. < 0.05	−5.75143673477	VL ($M_X < M_Y$)
			O3			
1	166.0	24	0.00230650528542	sig. < 0.05	0.159444506344	S ($M_X > M_Y$)
5	226.0	24	0.0475938676102	sig. < 0.05	−0.540498402677	L ($M_X < M_Y$)
10	22.0	24	9.17775908636^{-09}	sig. < 0.05	−2.19305668743	VL ($M_X < M_Y$)
15	25.0	24	1.28383893526^{-08}	sig. < 0.05	−1.91953699489	VL ($M_X < M_Y$)
20	0.0	24	7.07828112425^{-10}	sig. < 0.05	−5.07867480828	VL ($M_X < M_Y$)

Table 7.4 – Mann-Whitney U test results for CARDINAL vs Optimal-Static of Objectives O1,O2,O3 over 25 iterations with Cohen d's difference and effect size.

7.3.2.1 Generalised Immunisation Rate (O1)

The immunisation rate evaluation (O1) score balances the quantity of detectors with time taken to distribute those detectors (described in section 6.9). Scores range from positive (better than) to negative (worse than), where 0 is equal to SendAll's median performance.

On the larger network tests (n=[10,15,20]) we find that the engineered algorithm (Optimal-Static (OS)) consistently outperforms the others with median values above 0 in Table 7.3 and as plotted in Figure E.3a. Optimal-Static is 2nd in the n=5 network size, giving ranks [3rd,2nd,1st,1st,1st]. The ranks for the immune system inspired approach (Vanilla) were [2nd,3rd,3rd,3rd,3rd] over each test network size (n=[1,5,10,15,20]).

The spread of results for n=[10,15,20] by Optimal-Static and SendAll (SA) is monotonically increasing, where OS has much smaller standard deviations. Vanilla's spread for n=[5,10,15,20] fluctuates in the same pattern as Vanilla's *Dist-M1* standard deviation (and median) score showing its influence on the evaluation score. By comparison Vanilla (as is shown by the raw metric standard deviation results of *Dist-M1* and *DataSent-M3*) implicates lower performance reliability than the engineered (OS) and reference (SA) algorithms.

The Mann-Whitney U test results on immunisation rate (O1) data in Table 7.4 show significant difference between the AIS and engineered algorithms on sizes n=1,10,15,20. The 5 network node size results show a medium (M) effect size from Cohen's d equation in favour of Optimal-Static (given by its larger mean O1 value), but with no significant difference ($p = 0.08$) from the Mann-Whitney test.

At n=1, a small (S) effect is implicated in favour of Vanilla, with a significant P value. In virtual network sizes 10,15 and 20 the results show very large (VL) effect sizes and significant P values from each statistical tests, in favour of Optimal-Static.

7.3.2.2 Immunisation Rate for Low-Throughput Networks (O2)

The immunisation rate for low-throughput networks (O2) score, such as Supervisory Control and Data Acquisition (SCADA) networks, incorporates an unweighted amount of data transmitted along with the generalised immunisation rate (O1) performance scoring. Scores range from positive (better than) to negative (worse than), where 0 is equal to SendAll's median performance.

On the networked tests (n=[5,10,15,20]) we find that the engineered algorithm (Optimal-Static (OS)) shows the best O2 median values in Table 7.3 and as plotted in Figure 7.5b, giving ranks [3rd,1st,1st,1st,1st]. The AIS approach (Vanilla) is consistently ranked as [2nd,2nd,2nd,2nd,2nd], over all test network sizes by its median score.

NoNet's median results at n=20 show improved evaluated performance over the reference (SendAll). This is a false indicator, in that NoNet fails to distribute any detectors for $n > 1$. As explained above, we choose to exclude this algorithm's performance results.

The AIS algorithm's median performance shows a monotonic decrease in n=[5, 15, 20] by comparison to the reference, while from n=5 (0.313) to n=10 (0.33) is close to stable. The engineered algorithm's median performance increases over n=[5,10,15,20] by comparison to the reference.

Generally speaking, the spread from O1 to O2 has corresponding ranks, whose values are adjusted mildly by the addition of the data metric component.

The Mann-Whitney U test results on immunisation rate for SCADA networks (O2) data in Table 7.4 show significant difference between the AIS and engineered algorithms on sizes n=1,5,10,15,20.

The n=1 network size results show a small (S) effect size from Cohen's d equation in favour of Vanilla (indicated by $d > 0$ in this case) with $P = 0.01$. In virtual network sizes n=[5,10,15,20] a very large (VL) Cohen d effect size is implicated toward Optimal-Static, with consistently significant Mann-Whitney U test results. The greatest significance (smallest P-values) are found at n=10 and n=20 in favour of Optimal-Static.

7.3.2.3 Immunisation Rate for High-Throughput Networks (O3)

The immunisation rate for high-throughput networks (O3) (such as enterprise networks) score incorporates the (lowly weighted) amount of data sent with the immunisation rate (O1) (as described in section 6.9). Scores range from positive (better than) to negative (worse than), where 0 is equal to SendAll's median performance.

On the network tests n=[10,15,20] we find that the engineered algorithm (Optimal-Static (OS)) gives the best O3 median values in Table 7.3 and as plotted in Figure E.3b. Based upon medians OS ranks as [3rd,2nd,1st,1st,1st] for n=[1,5,10,15,20]. The AIS approach (Vanilla) ranks on median values at [2nd, 3rd, 3rd, 3rd, 3rd]. The SendAll algorithm performs most favourably in n=5 and second most in n=[10,15,20].

The AIS algorithm's median performance shows a monotonic decrease in n=[5, 15, 20] by comparison to the reference, while from n=5 (-0.054) to n=10 (-0.039) is close to stable. The engineered algorithm's median performance increases over n=[5,10,15,20]. The standard deviation results from O2 to O3 corresponds similarly with those from O1 to O2.

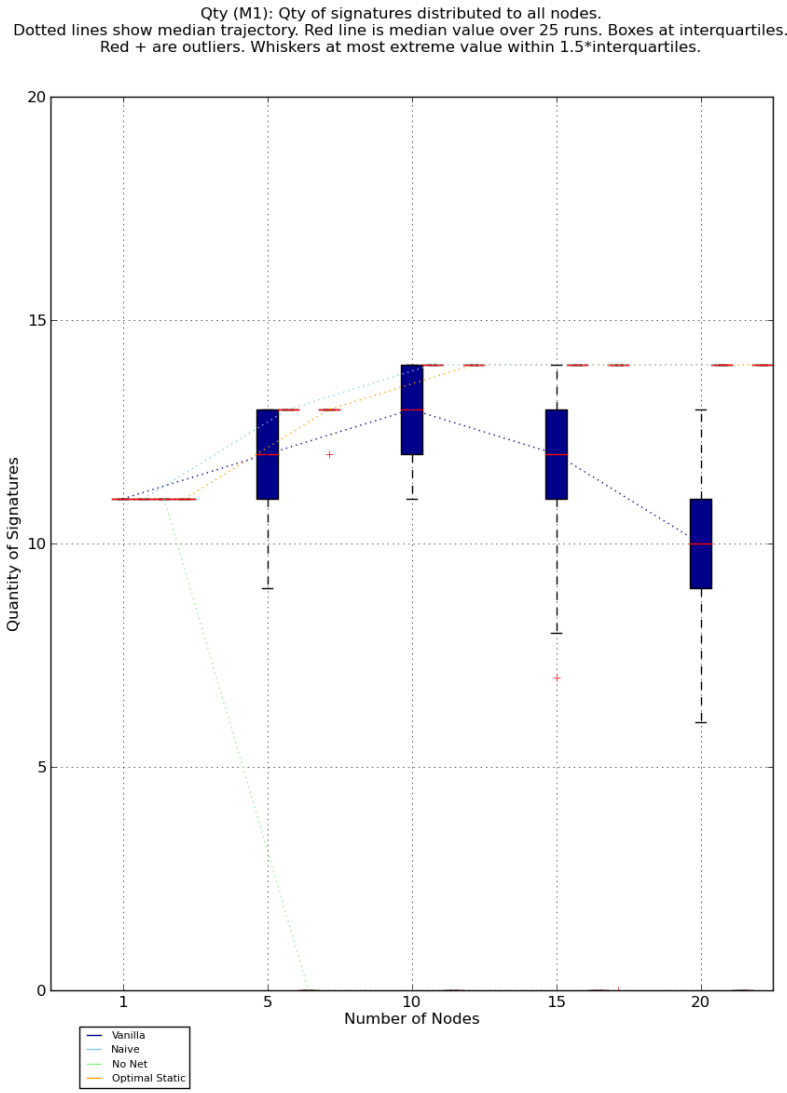
The Mann-Whitney U test results on immunisation rate for enterprise networks (O3) data in Table 7.4 show significant difference between the AIS and engineered algorithms on sizes n=1,5,10,15,20.

The n=1 network size results show a small (S) effect size from Cohen's d equation in favour of Vanilla (indicated by $d > 0$) with $P = 0.002$. In virtual network sizes n=[5] a large (L) effect, and in n=[10,15,20] a very large (VL) Cohen d effect size is implicated in favour of Optimal-Static, with consistently significant Mann-Whitney U test results. The greatest significance (smallest P-values) are found at n=10 and n=20 in favour of Optimal-Static.

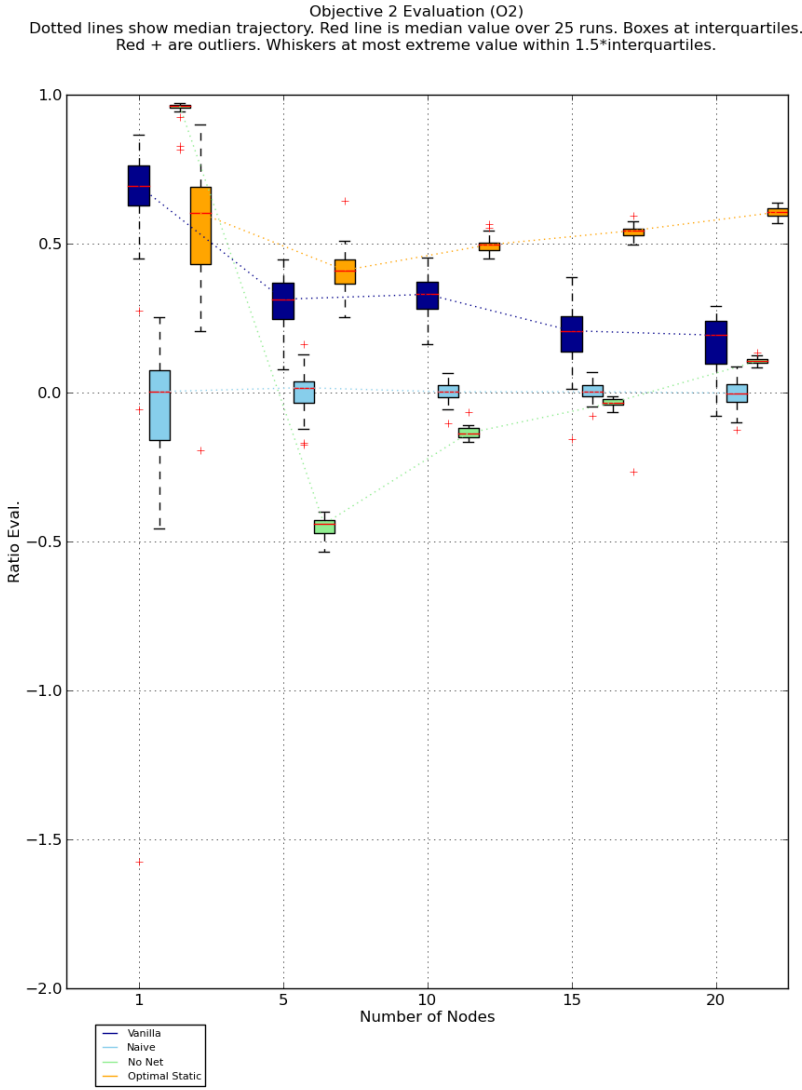
7.3.3 Further Discussion

- The memory footprint required by Optimal-Static's algorithm for a host to maintain the detector states of each neighbouring host is likely to be smaller than the ongoing transmissions to request a minimal hashed representation of the detector state. Of course in a worse case scenario, Optimal-Static's algorithm will fail to distribute optimally. A comparison to this alternative engineered request algorithm remains as open work.

- Because CARDINAL-Vanilla's probability-based priority heuristic and detector selection mechanism gives lower priority to rare inputs, it is a poor representation for systems where a single malicious input can cause as much damage as many inputs.



(a) Metric observations. Number of detectors distributed during experiment.



(b) Immunisation rate suitability upon SCADA networks (O2), combining metric M1,M2,M3 data using our extension of the Weisbin & Rodriguez ratio equation with SendAll as reference.

Figure 7.5 – Box plot observations of detector distribution quantity metric and multi-objective evaluation (O2) on the virtual network tests.

- The underpinning decision that defines CARDINAL-Vanilla's *under-attack* condition is a poor representation for a detection system that must respond to single malicious inputs, see Equation 5.7 in section 5.10.1. This is because CARDINAL's T_{UA} threshold approach (set at 50) is set too high for the values given by our simple binary output classifier, see 5.7.1. Therefore the system is biased to a slow response.

- The experiment methodology represents a model test scenario for defence against malicious input-based network attacks. However, it appears that the arbitrarily selected signatures required for *Dist-M1* and described in 6.2.2, are not perfectly suited to the *under-attack* condition or the probability-based selection mechanism due to the relatively low occurrence of those signatures in the dataset. This leads to sub-optimal actions taken by CARDINAL-Vanilla. The specific probability depends on the order and repetition of inputs from the dataset. This is different within each dataset. Without adding an adaptive mechanism based on the incoming data, the condition and selection mechanism will perform sub-optimally.

7.3.4 Conclusions

This study achieved the first benchmark results of a *fair* implementation of the *CARDINAL* model by (Kim *et al.* , 2005) and found some test conditions that support CARDINAL-Vanilla's mechanism for selection and distribution of detectors by comparison to two benchmark and one engineered algorithmic approach. A new validation methodology has been used to evaluate the architecture as a distributed self-healing system and has enabled a comparative use case study of two underlying architecture algorithms. In addition, the first benchmark parameter configuration set has been presented and tested for the CARDINAL model.

Under these test conditions we discovered the *CARDINAL* self-healing system's immunisation rates (i.e. O2, O3) performed similarly in small network sizes however as the virtual networks size increased the engineered distribution selection mechanism outperformed the bio-inspired solution.

In fact, according to the detector distribution *Dist-M1* results we can identify a salient performance degradation as the network sizes increase. We found a significant correlation between the time delay and ordered node number which we thought may have caused this behaviour, however our analysis in 7.6 shows that the behaviour exists regardless of the time delay. We can reject other possible causes such as CARDINAL's probability-based (roulette wheel) selection of detectors and the linear % ordering of the distributed user behavioural model as these aspects are found in the following enterprise network benchmark tests but are without the presence of the degradation with respect to the network size. This leads us to believe this was an effect of process and I/O scheduling and sub-process/ thread execution order of the operating system scheduling behaviour. We can consider this as an operational noise factor or a performance bottleneck breach on the virtual network. Either way, we find this the most important performance discrepancy between the virtual and enterprise tests of the architecture.

The performance reliability of *CARDINAL*-Vanilla as shown by the standard deviation over the metrics and system measures varies more than the other engineered and naïve algorithms. We speculated that this larger variability is caused by the probability selection of detectors and destination hosts or the voluminosity-based priority heuristic. However, the enterprise benchmark tests that follow show this effect is no longer noticeable. Thus we conclude this to be a factor of uncontrolled noise on the

virtual network and one that may be noticeable as the network size expands to its bottleneck capacity threshold point.

7.3.5 Next Steps

In our next work we shall repeat the test in a real network setting and undertake a search for best performing (optimal) parameter configuration. This study evaluated one configuration of CARDINAL-Vanilla's AIS parameters, at this stage we are unaware of whether the architecture will perform better with a different parameter configuration set. Therefore an analysis of a range of parameter value sets is another item of evaluation of the architecture presented later in the thesis.

7.4 Comparison on Enterprise Networks

7.4.1 Objectives

The earlier benchmark experiment gave immunisation rate results of our CARDINAL-Vanilla implementation as well as our engineered algorithm within a virtual network environment. This next series of experiments will lend its focus to retesting that set-up on an enterprise network of computers. We shall repeat the previous test comparing the CARDINAL-Vanilla and Optimal-Static algorithms for distribution, as described in 7.2.2, under an enterprise network of connected computers.

7.4.2 Experiment Design

In this section we state the changes to the experiment design and validation methodology from the earlier virtualised network tests.

As the enterprise network on which these tests run is in daily operation, we reduce the quantity of runs from 25 to 10 and increase the maximum number of architecture nodes to match our 41 computer limit. The instance addressing is updated from sequential TCP port numbers, to IP host name addresses (on 193.x.x.x) receiving stream data over TCP on port 60175. Each client has a new pre-built set of neighbouring client addresses. Note that we do not compress or encrypt the communications at this stage as it does not assist our current benchmarking objective. Its effect upon the measures would increase the data sent and processing time in a uniform manner.

Many of the virtualised test components and configurations are directly cloned to these enterprise network experiments. Among the unchanged test components, are the single segment network topology, shown diagrammatically in Figure 7.2, the labelled CSIC v0.5.2 dataset described in Table 6.8 and the experiment learning phases. Also unchanged is the experiment procedure described in 6.4.1, the *CARDINAL* architecture implementation, its parameter configuration and the heterogeneous user behavioural model described in 6.6.

The following sections will describe the changes and implications of those changes from the earlier virtual network tests.

7.4.2.1 Measurements

Measurement data of each architecture node is logged on the local machine to avoid impacting network traffic and to reduce scaling effects (i.e. exponential transmissions and I/O waiting times) during trials. After a set of trials has completed, the individual experiment logs created by each instance are collated and parsed to extract global measurements (metrics).

We have described the metrics in the previous experiment (and in detail in 6.2), these are detector distribution quantity (*Dist-M1*), median time to distribute a detector (*Time-M2*) and data required to distribute all detectors (*DataSent-M3*). We continue to evaluate each experiment's performance according to our three objectives: immunisation rate (O1), immunisation rate on low capacity networks (O2) and on high capacity networks (O3) as discussed in 6.9.

The log collection locations and parsing location now differ in order to manage the extra data. As the maximum quantity of our nodes has more than doubled, our log lengths have roughly doubled and their quantity has doubled again. This has no additional effect on the extracted measurement values. Each node logs to memory during the trials and writes to file as soon as a run concludes; a process

that remains unchanged from the virtual network experiment set-up. After a set of iterations the logs are compressed and transferred to one location (at offset times). At a later point a single machine takes over to uncompress, combine and then parse the logs to extract the global metrics as described in 7.4.3.2.

7.4.3 Configuration

7.4.3.1 Environment & Preparation

The experiment network tests ran on a single network segment, consisting of 41 machines and a network switch. All 41 machines ran during each trial, however only n machines executed the experimental code. Trials ran with $n = 1, 5, 10, 15, 20, 30$ and 41 machine quantities. A summary of the environment configuration is in D.2.

Before the experiment begins each workstation is issued with a network time protocol (NTP) update (`ntpdate`); their timestamp values are then manually confirmed to match (have a average range less than 500ms). This happens by the node echoing its timestamp upon receipt of a Secure Shell (ssh) (remote login and command execution) request. These ssh requests are issued as background processes. This leaves an amount of imprecision equivalent to time to issue and execute a background ssh connection process in bash. Improving upon this time synchronization precision remains open to further research.

The trials were run while other users were logged out (and had no access to) the computers. Prior to an experiment run steps were taken to ensure that no computers were exorbitantly using their resources (where `/proc/loadavg` reported 0.05 or less over the past 5 minutes of usage). Processes that exceeded this minimum were either remotely terminated or their operating system was rebooted. However, as with the virtual network tests, we cannot exclude the possibility of noise effects from other running processes or operating system and package updates. In this enterprise network of 41 machines the effect, if any, is in likelihood greater than that of the virtual network tests running on a single machine. We believe we have achieved a fair and representative state of a real network with low load; as such further analysis of this potential effect remains out of the scope of our current work.

7.4.3.2 Execution Script

Two GNU bash scripts are executed. The first is an overall *experiment control* script which is run on a networked machine exclusive (external) to the test workstations.

The second *deployment, run and upload* script is copied to each of the n machines at the beginning of each test and executed locally. This script downloads and deploys the experiment code to the local disk (`/tmp`), waits until the issued time, then runs the trial code. Once a trial is completed, the local script compresses and uploads the compressed log files to a network location. The final node's deployment script then waits until all logs have been uploaded and then downloads, un-compresses and parses those logs for the global metrics. Network downloads and uploads required by multiple nodes are executed in a staggered (node-number dependent) time manner to avoid denial of service to other enterprise network infrastructure.

The default resource limits (`ulimit`) required no increase during the network tests, as each computer executed only one instance of the CARDINAL client application. The number of threads per *CARDINAL* instance was $9 + (n * 4)$), remaining unchanged from the previous virtual configuration

(see 7.2.4).

Once again, the Java code was run within the Java virtual machine (JVM) version 1.7 in 64bit mode with the argument -d64. We set an equal maximum amount of allocated heap space memory per instance at 1024 megabytes during all network test sizes – which, by comparison, differs only at $n = 20$ where in the earlier virtual tests the max heap size was 793.6MiB. We do not expect this difference had an effect on our metric measures.

To match the earlier virtual test configuration, the STDERR and STDOUT streams were disabled and experiment logging was buffered into memory and output to file at the end of each experiment. This leads to an exponential scaling factor on the memory footprint size with respect to input data size as in 7.2.4.

7.4.3.3 Network Hosts Configuration

Each node had a unique network configuration file and separate from the *CARDINAL-Vanilla* configuration. Firstly this contained a set of neighbouring IP addresses. For example in the five node network test, the first node had four neighbouring IP addresses as 193.x.x.2:60175 to 193.x.x.5:60175.

This configuration file also contained the parameter values of the distributed user model specific for this node, as detailed in 6.6 and the experiment constants defined in 6.3. The noteworthy user specific model parameters are the random seed $\mathbb{M}_{\mathbb{STAT}s}$ – which feeds the dataset input selection in the user model and the roulette wheel selection mechanism within CARDINAL-Vanilla – and the anomalous proportion $\mathbb{M}_{\mathbb{STAT}a_n}$ that drives the learning.

7.5 Results

We present the metric scores and immunisation rate performance scores of the two algorithms. Table 7.5 contains the raw metric results and Table 7.7 contains the combined immunisation rate scores. Medians (η), interquartile ranges (IQR) and standard deviations *(Std)* over 10 trials are presented over the seven enterprise network sizes (n) of 1,5,10,15,20,30 and 41 workstations.

In tables 7.6 and 7.8 we present the difference significance results between the two algorithms. We use Mann-Whitney U statistical tests to measure difference between the AIS algorithm (Vanilla) and the engineered algorithm (Optimal-Static). The Cohen's d effect size is presented to give a measure of difference scale. The trial run scores of the two algorithms, per network size are inputs into the equations. The inferential statistical test selection process follows that of the previous virtual network tests, where the test and its results' complete description can be found in section E.1 and the Mann-Whitney U test in subsection E.1.1.

7.5.1 Metric Performance

The following three subsections will report the performance of the algorithms on the metrics *Dist-M1*, *Time-M2* and *DataSent-M3*.

7.5.1.1 Number of Detectors Distributed to Entire Network (M1)

The *Dist-M1* score here has a theoretical maximum of 20, as defined by the number of instances in (`arbitrary_signatures.csv`) the accompanying signatures dataset file. Within the 9% sample anomalous dataset 17 of these instances reside used within these tests. However, 14 is our observed maximum quantity of matches that. At network sizes 1,5,10,15,20,30 and 41 the observed maximum *Dist-M1* values are 11,13,14,14,14,14 and 14.

All network test sizes show equal (maximum) median *Dist-M1* results for Optimal-Static and Vanilla. Optimal-Static shows zero result dispersion over the 10 iterations, as both the interquartile range (IQR) and standard deviation show 0.0 on all tests. The Artificial Immune System (AIS) algorithm has an IQR of 1.0 at n=5, whereas all other network sizes show IQR equal to zero. Standard deviation reports the Vanilla algorithm to have variation at n=5,30 and 41, caused by iterations falling outside of the 25% and 75% interquartile ranges. Figure 7.9a shows zero variation among Vanilla iterations at n=10,15 and 20. At n=30 and n=41, two and one (respective) iteration results show values less than the median. A Pearson's correlation coefficient test between both algorithms shows a significant linear correlation with a P-value of 1.0 at all network sizes.

Figure 7.6 shows some similarity to the trajectory of the virtualised network trial results found earlier. However, the "drop-off" begins at an unmatched (larger, n=30) network size in the enterprise tests and the standard deviations are evidently smaller in these enterprise network trials. If we calculate the standard deviations for only the first ten trials from each virtualised test we have [0.823,1.059,1.969,2.331] for n=[5,10,15,20]. Such that, each network size's standard deviation is comparatively larger in the virtualised tests than the enterprise network tests over an equal number of trials. Why this result discrepancy occurs between the two network types remains a subject of inquiry.

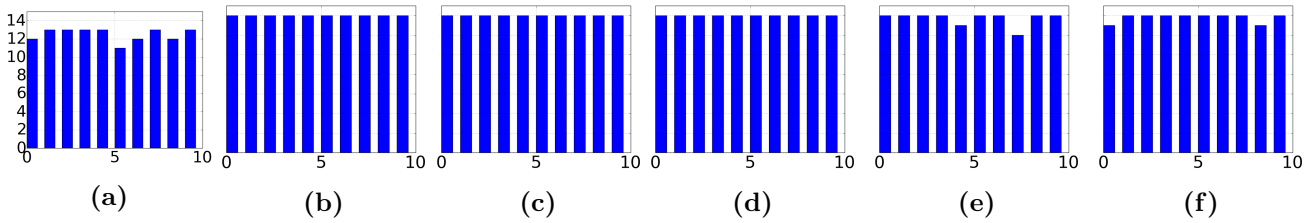


Figure 7.6 – *Dist-M1* variation of Vanilla performance over 10 runs with **(a)** 5, **(b)** 10, **(c)** 15, **(d)** 20, **(e)** 30 and **(f)** 41 nodes. Y-axis shows quantity of detectors (M1) from 0 to 14 (max), X-axis shows the test run number from 1 to 10.

The Mann-Whitney U test results for difference between Vanilla and Optimal-Static on *Dist-M1* data are in Table 7.5, showing significant difference only at network size 5, where the critical probability value is 0.05 and very large (VL) effect sizes in favour of Optimal-Static over Vanilla. The performance difference at all remaining network sizes was insignificant. At sizes n=30 and n=41, a large effect in Cohen's d equation is shown in favour of Optimal-Static, however the P-value difference is shown to be insignificant with $p \sim= 0.08$.

7.5.1.2 Time Taken to Distribute Median Detector (M2)

The *Time-M2* score is the measured time duration between a detector being first classified and the time at which it has been distributed to all other nodes on the network. Therefore values must be non-negative and smaller values are preferred, as discussed in section 6.2.

The best median *Time-M2* time durations were given by Vanilla for n=[1,5,10] and by Optimal-Static for n=[15,20,30,41]; implying faster distribution on larger networks by the engineered algorithm. Figure E.4 visually reiterates the relatively low interquartile ranges from both algorithms as found in Table 7.5. Also shown is an outlier at n=10 for the Optimal-Static algorithm leading to its high standard deviation for that network size.

A pattern of apparent uncontrolled noise is recognisable within the interquartile ranges and standard deviations for both algorithms. For example, at n=20 the Vanilla and Optimal-Static algorithms gives an IQR difference of 2.151 (2.658 and 0.507); whereas that difference is 0.013 (1.072 and 1.085) at n=15 and 0.198 (1.709 and 1.511) at n=30. By design, the IQR will remove the effect of outliers, but does not remove the effect of ongoing noise or the test's intended behaviour. This result dispersion may have shown signs of settling within a more controlled enterprise network environment, however we did not have access to an experimental computer network of such a size.

The virtualised tests clearly show higher median times by comparison to the enterprise tests for each of the networked tests as a result of the number of processors available to the architectures.

The Mann-Whitney U test results for difference between Vanilla and Optimal-Static on *Time-M2* (time taken) data are in Table 7.5, showing significant difference at n=10 with medium effect size from Cohen's d equation in favour of Vanilla. An increasing significant difference (P-value decreases at n increases) with very large effect size is reported at n=[15,20,30,41] in favour of Optimal-Static.

The box plots in Figure 7.7 show that the difference between the two algorithms, while significant, are relatively similar over all of the network sizes.

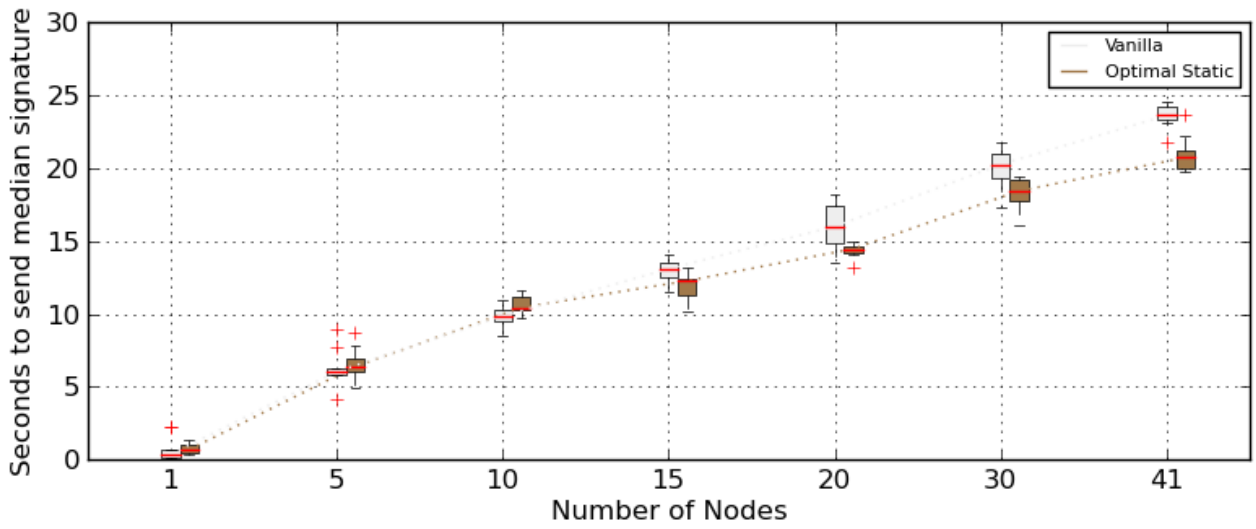


Figure 7.7 – Box plot showing the relative difference in *Time-M2* metric scores for the AIS and engineered algorithms for selection and distribution of detector modules.

7.5.1.3 Total Data Sent During Experiment to Distribute (M3)

The *DataSent-M3* score shows the total amount of data sent in megabytes (MiB) by all nodes during an experiment iteration. As with time taken, the values are non-negative, smaller values are preferred and a full discussion can be found in section 6.2.

Table 7.5 shows Vanilla's *DataSent-M3* median results are consistently worse than the engineered algorithm, as plotted in Figure E.5a. The percentages of Optimal-Static's data sent (numerator) to Vanilla's (denominator) are [28.6%, 34.7%, 42.5%, 49.5%, 52.3%, 57.3% and 57.7%] over each n size. The interquartile ranges from Optimal-Static are monotonically increasing with n, the same goes for Vanilla at n=[1,5,15,20,30,41] where n=10 is an exception. At each n size Vanilla shows larger variation in its interquartile ranges and standard deviations compared to the engineered algorithm.

We can extrapolate a "per test, per machine variation" for both algorithms as IQR/n, giving [0.23,0.41,0.23,0.54,0.37 and 0.38] for Vanilla and [0,0,0.02,0.05,0.1 and 0.19] for Optimal-Static over n=[5,10,15,20,30 and 41]. This again shows larger (extrapolated) variation per machine in Vanilla than Optimal-Static.

The Mann-Whitney U test results for difference between Vanilla and Optimal-Static on *DataSent-M3* data sent are in Table 7.5, showing significant and very large Cohen's d effect size at all network sizes. A monotonic increase in significance is recognisable from n=1 to n=20, suggesting the difference exacerbates (comparatively, Vanilla becomes worse where Optimal-Static improves) as the network size increases.

Figure 7.8 shows a box plot representation of the differences between the two algorithms. We can clearly see that the quantity of data sent is different and in cases is transmitting roughly double the data quantities overall; which in its current state for application on SCADA networks, is unacceptable.

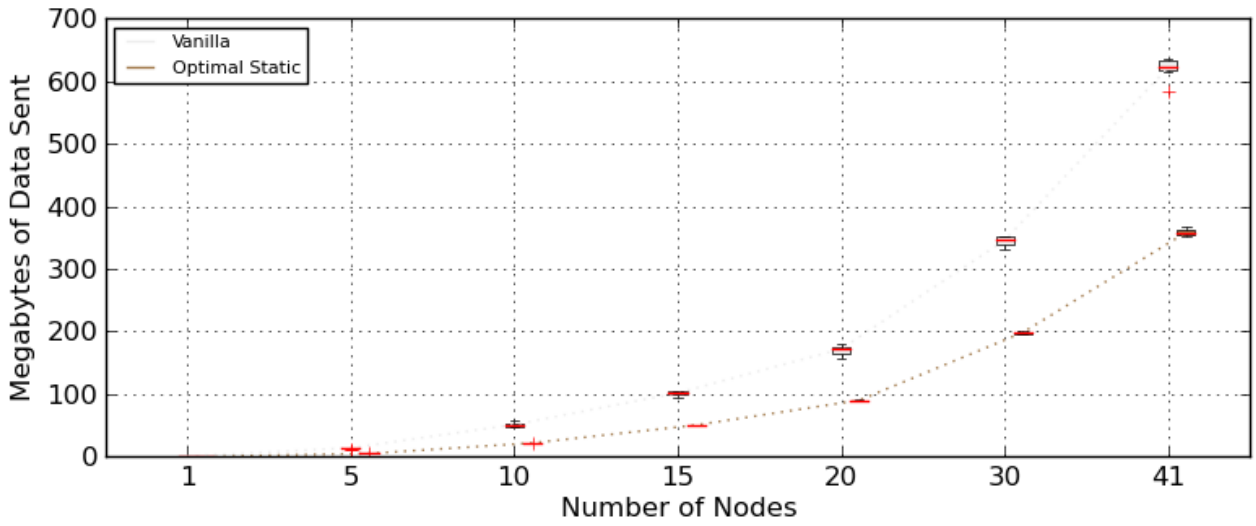


Figure 7.8 – Box plot showing the relative difference in *DataSent-M3* metric scores for the AIS and engineered algorithms for selection and distribution of detector modules measured over each trial.

n	η	Vanilla (IQR)	Vanilla (Std)	η	Optimal-Static (IQR)	Optimal-Static (Std)
			Dist-M1			
1	11.0	(0.0)	(0.0)	11.0	(0.0)	(0.0)
5	13.0	(1.0)	(0.671)	13.0	(0.0)	(0.0)
10	14.0	(0.0)	(0.0)	14.0	(0.0)	(0.0)
15	14.0	(0.0)	(0.0)	14.0	(0.0)	(0.0)
20	14.0	(0.0)	(0.0)	14.0	(0.0)	(0.0)
30	14.0	(0.0)	(0.64)	14.0	(0.0)	(0.0)
41	14.0	(0.0)	(0.4)	14.0	(0.0)	(0.0)
			Time-M2			
1	0.368	(0.535)	(0.778)	0.683	(0.527)	(0.331)
5	6.042	(0.34)	(1.193)	6.386	(0.899)	(1.027)
10	9.853	(0.826)	(0.711)	10.441	(0.839)	(35.694)
15	13.099	(1.072)	(0.833)	12.268	(1.085)	(0.839)
20	16.005	(2.658)	(1.548)	14.468	(0.507)	(0.48)
30	20.259	(1.709)	(1.299)	18.438	(1.511)	(1.0)
41	23.622	(0.869)	(0.753)	20.724	(1.316)	(1.183)
			DataSent-M3			
1	0.7	(0.1)	(0.049)	0.2	(0.0)	(0.0)
5	13.85	(1.15)	(1.544)	4.8	(0.0)	(0.03)
10	51.05	(4.125)	(2.804)	21.7	(0.075)	(0.078)
15	101.65	(3.375)	(2.775)	50.3	(0.375)	(0.219)
20	171.95	(10.725)	(7.101)	90.0	(1.05)	(0.808)
30	345.6	(11.05)	(6.931)	198.2	(2.975)	(1.555)
41	621.35	(15.675)	(14.784)	358.25	(7.7)	(4.765)

Table 7.5 – Tabular results of metrics M1,M2,M3 showing medians (η), interquartile ranges (IQR) (75%-25%) and standard deviations (Std) (with n-1 denominator) over 10 iterations of each algorithm. Values are rounded to 3 significant digits.

n	U	df	P	**Significance**	**Cohen's d**	**Effect Size**
			Dist-M1			
1	Nan	9	$Invalid$	insig.	0.0	None
5	30.0	9	0.017212	sig. < 0.05	-1.0	VL ($M_X < M_Y$)
10	Nan	9	$Invalid$	insig.	0.0	None
15	Nan	9	$Invalid$	insig.	0.0	None
20	Nan	9	$Invalid$	insig.	0.0	None
30	40.0	9	0.084039	insig.	-0.628587	L ($M_X < M_Y$)
41	40.0	9	0.083744	insig.	-0.67082	L ($M_X < M_Y$)
			Time-M2			
1	30.0	9	0.069859	insig.	-0.051407	S ($M_X < M_Y$)
5	36.0	9	0.153745	insig.	-0.259625	S ($M_X < M_Y$)
10	22.0	9	0.018818	sig. < 0.05	-0.473186	M ($M_X < M_Y$)
15	19.0	9	0.010567	sig. < 0.05	1.095983	VL ($M_X > M_Y$)
20	18.0	9	0.008629	sig. < 0.05	1.447793	VL ($M_X > M_Y$)
30	18.0	9	0.008629	sig. < 0.05	1.393088	VL ($M_X > M_Y$)
41	7.0	9	0.000657	sig. < 0.05	2.570759	VL ($M_X > M_Y$)
			DataSent-M3			
1	0.0	9	2.3^{-05}	sig. < 0.05	12.597619	VL ($M_X > M_Y$)
5	0.0	9	4.3^{-05}	sig. < 0.05	7.169698	VL ($M_X > M_Y$)
10	0.0	9	7.4^{-05}	sig. < 0.05	13.858142	VL ($M_X > M_Y$)
15	0.0	9	8.6^{-05}	sig. < 0.05	24.644184	VL ($M_X > M_Y$)
20	0.0	9	9.1^{-05}	sig. < 0.05	15.10875	VL ($M_X > M_Y$)
30	0.0	9	9.1^{-05}	sig. < 0.05	27.657829	VL ($M_X > M_Y$)
41	0.0	9	9.1^{-05}	sig. < 0.05	22.651728	VL ($M_X > M_Y$)

Table 7.6 – Mann-Whitney U test results for CARDINAL vs OptimalStatic of Metrics M1,M2,M3 over 10 iterations with Cohen d's difference and effect size. An *Invalid* value in the P column and a *Nan* in the U statistic column represents identical results on all iterations of both algorithms in the respective Mann-Whitney U test.

	Vanilla			Optimal-Static		
n	η	**(IQR)**	**(Std)**	η	**(IQR)**	**(Std)**
			O1			
1	0.231	(0.391)	(0.569)	0.0	(0.386)	(0.242)
5	0.016	(0.053)	(0.111)	0.0	(0.07)	(0.08)
10	0.028	(0.04)	(0.034)	−0.0	(0.04)	(1.709)
15	−0.034	(0.044)	(0.034)	0.0	(0.044)	(0.034)
20	−0.053	(0.092)	(0.053)	0.0	(0.018)	(0.017)
30	−0.058	(0.05)	(0.047)	0.0	(0.041)	(0.027)
41	−0.082	(0.022)	(0.021)	0.0	(0.032)	(0.029)
			O2			
1	−1.019	(0.641)	(0.649)	0.0	(0.386)	(0.242)
5	−0.912	(0.181)	(0.098)	0.0	(0.07)	(0.081)
10	−0.645	(0.093)	(0.067)	−0.001	(0.038)	(1.709)
15	−0.552	(0.065)	(0.048)	−0.001	(0.046)	(0.034)
20	−0.483	(0.104)	(0.059)	−0.0	(0.019)	(0.018)
30	−0.421	(0.079)	(0.05)	0.005	(0.035)	(0.026)
41	−0.458	(0.045)	(0.035)	0.005	(0.023)	(0.025)
			O3			
1	0.106	(0.416)	(0.577)	0.0	(0.386)	(0.242)
5	−0.078	(0.054)	(0.099)	0.0	(0.07)	(0.08)
10	−0.037	(0.036)	(0.034)	−0.0	(0.04)	(1.709)
15	−0.086	(0.044)	(0.035)	−0.0	(0.044)	(0.034)
20	−0.099	(0.095)	(0.053)	−0.0	(0.018)	(0.017)
30	−0.095	(0.055)	(0.047)	0.001	(0.041)	(0.027)
41	−0.12	(0.027)	(0.022)	0.001	(0.031)	(0.028)

Table 7.7 – Tabular results of objectives O1,O2,O3 showing medians (η), interquartile ranges (IQR) (75%-25%) and standard deviations (Std) (with n-1 denominator) over 10 iterations of the Vanilla and Optimal-Static algorithms. Values are rounded to 3 significant digits.

n	U	df	P	**Significance**	**Cohen's d**	**Effect Size**
			O1			
1	30.0	9	0.069859	insig.	0.051407	S ($M_X > M_Y$)
5	42.0	9	0.285375	insig.	0.045085	S ($M_X > M_Y$)
10	22.0	9	0.018818	sig. < 0.05	0.473186	M ($M_X > M_Y$)
15	19.0	9	0.010567	sig. < 0.05	−1.095983	VL ($M_X < M_Y$)
20	18.0	9	0.008629	sig. < 0.05	−1.447793	VL ($M_X < M_Y$)
30	18.0	9	0.008629	sig. < 0.05	−1.40588	VL ($M_X < M_Y$)
41	5.0	9	0.000384	sig. < 0.05	−2.715066	VL ($M_X < M_Y$)
			O2			
1	0.0	9	8.9^{-05}	sig. < 0.05	−2.180835	VL ($M_X < M_Y$)
5	0.0	9	9.1^{-05}	sig. < 0.05	−9.024596	VL ($M_X < M_Y$)
10	10.0	9	0.001414	sig. < 0.05	−0.05062	S ($M_X < M_Y$)
15	0.0	9	9.1^{-05}	sig. < 0.05	−12.513134	VL ($M_X < M_Y$)
20	0.0	9	9.1^{-05}	sig. < 0.05	−11.026788	VL ($M_X < M_Y$)
30	0.0	9	9.1^{-05}	sig. < 0.05	−10.071471	VL ($M_X < M_Y$)
41	0.0	9	9.1^{-05}	sig. < 0.05	−13.591509	VL ($M_X < M_Y$)
			O3			
1	42.0	9	0.285012	insig.	−0.195883	S ($M_X < M_Y$)
5	23.0	9	0.022577	sig. < 0.05	−0.856515	VL ($M_X < M_Y$)
10	34.0	9	0.120661	insig.	0.420812	M ($M_X > M_Y$)
15	2.0	9	0.000165	sig. < 0.05	−2.488031	VL ($M_X < M_Y$)
20	0.0	9	9.1^{-05}	sig. < 0.05	−2.549087	VL ($M_X < M_Y$)
30	4.0	9	0.000291	sig. < 0.05	−2.319322	VL ($M_X < M_Y$)
41	1.0	9	0.000123	sig. < 0.05	−4.067139	VL ($M_X < M_Y$)

Table 7.8 – Mann-Whitney U test results for CARDINAL vs OptimalStatic of Objectives O1,O2,O3 over 10 iterations with Cohen d's difference and effect size.

7.5.2 Immunisation Rate System Performance

The following three system performance measures O1,O2 and O3 (as described in 6.9) use a ratio of differences equation (specified in 6.10.3) to calculate each algorithm's performance. The Optimal-Static algorithm is used as our reference system. Differences between Vanilla's median result to Optimal-Static's median result can be compared. Generally speaking, Vanilla's scores can be greater (better than), less (worse than) or equal to Optimal-Static's median performance.
The evaluation results are shown in Table 7.7 and the significance test for difference between the algorithms is shown in Table 7.8.

7.5.2.1 Generalised Immunisation Rate (O1)

The immunisation rate evaluation (O1) score balances the quantity of detectors (*Dist-M1*) with time taken to distribute those detectors (*Time-M2*). This provides a broad but not very discriminating survey of performance.

Table 7.7 shows the median O1 results of Vanilla evaluated as better than the engineered algorithm during the n=[1,5,10] tests. Whereas at the larger n sizes (15,20,30 and 41) the reverse applies, with monotonically increasing median distances in favour of Optimal-Static. The interquartile range ranks suggest reliability to be network size-specific, yet IQR distances between both algorithms are similar. Per network size, Vanilla's IQR ranks are [2nd,1st,-,-,2nd,2nd,1st], where '-' represents equal to Optimal-Static.

The Mann-Whitney U test for difference between the AIS and engineered algorithms on immunisation rate (O1) data in Table 7.8 shows significant difference in network sizes n=10 to n=41. At n=10, Cohen's d equation suggests a medium effect size in favour of Vanilla, whereas at n=[15,20,30 and 41] a very large effect size is in favour of Optimal-Static. At n=1 and n=5 Vanilla is preferred with a small effect size and yet with insignificant P-values ($P \sim= 0.06$ and $P \sim= 0.28$ respectively).

The O1 immunisation rate results show Optimal-Static to be preferred with significant difference in performance at the larger network sizes (15,20,30 and 41) and the AIS algorithm to be preferred at network size 10, with insignificant improvement at the two smallest network sizes.

7.5.2.2 Immunisation Rate for Low-Throughput Networks (O2)

The immunisation rate for low-throughput networks (O2) score, such as SCADA networks, incorporates an unweighted amount of data transmitted along with the generalised immunisation rate (O1) performance scoring.

Table 7.7 shows Optimal-Static's median O2 results evaluated better than the Vanilla algorithm at all n network sizes. Interquartile ranges report higher variation from Vanilla over Optimal-Static at all sizes also. These points can be recognised visually in Figure 7.9b.

The results of the Mann-Whitney U statistical difference test between the AIS and engineered algorithms on immunisation rate for SCADA-like networks (O2) can be found in Table 7.8. Here we can identify strong significance ($p < 0.001$) and Cohen's d equation difference in favour of Optimal-Static at all network sizes n=[1,5,10,15,20,30 and 41]. The Cohen's d effect size is very large at all network n sizes except n=10, where it is reported as small.

The immunisation rate results for SCADA-like networks (O2) show Optimal-Static to be preferred with significant difference in performance on all tested network sizes (1,5,10,15,20,30 and 41 nodes) instead of the AIS algorithm.

7.5.2.3 Immunisation Rate for High-Throughput Networks (O3)

The immunisation rate for high-throughput networks (O3) (such as enterprise networks) score incorporates the (lowly weighted) amount of data sent with the immunisation rate (O1) (as described in section 6.9).

Table 7.7 shows Optimal-Static's median O3 results for all the networked sizes (5 through 41) evaluate as more suitable than Vanilla. However, Vanilla's O3 median suitability results are comparatively better (in all tests) than its O2 evaluation. Interquartile ranges report higher variation from Vanilla over Optimal-Static at n=[1,5,10,20 and 30], equal at 15 nodes and slightly less variation at 41 nodes.

The results of the Mann-Whitney U statistical difference test between Vanilla and Optimal-Static on immunisation rate for enterprise networks (O3) are in Table 7.8 with Cohen's d effect size. The Mann-Whitney U tests shows significant difference between the two algorithm at network sizes n=[5,15,20,30 and 41]. Cohen's d show that difference in favour of Optimal-Static, each with a very large (VL) effect size.

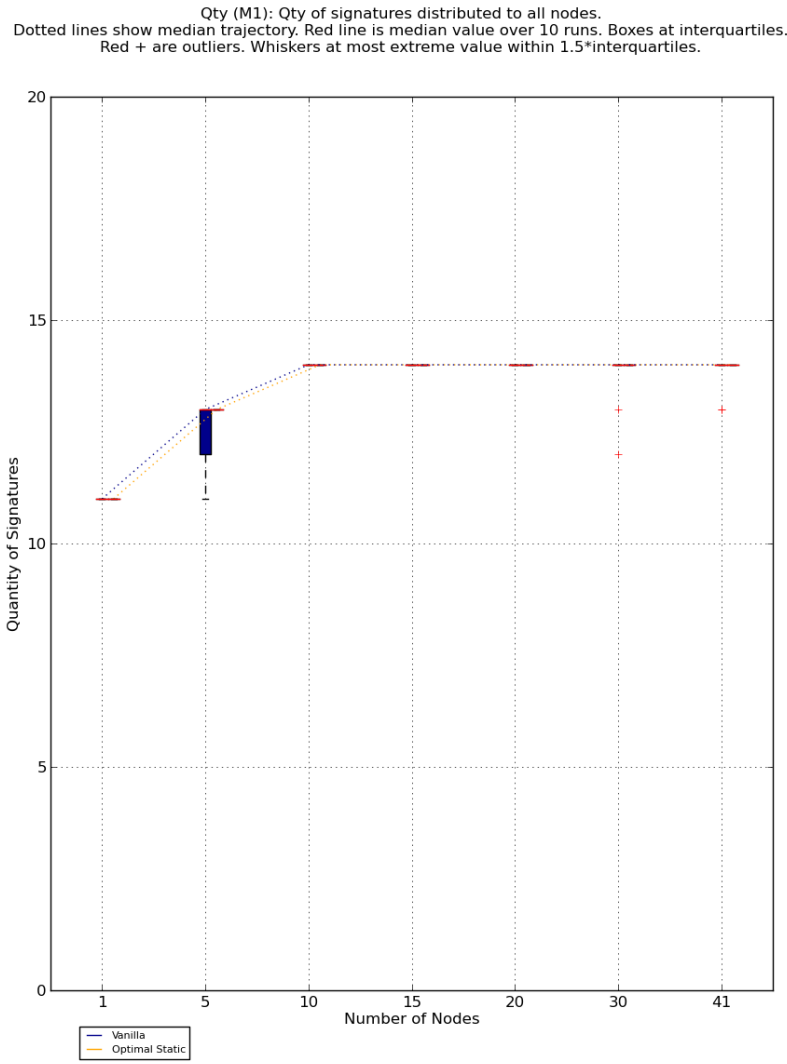
The immunisation rate results for enterprise networks (O3) show that Optimal-Static is evaluated better with a strong significant improvement ($P < 0.001$) on network sizes of 15,20,30 and 41 nodes instead of the Vanilla algorithm and a significant improvement ($P \sim= 0.02$) on the 5 node network size tests. Vanilla's improved performance on the 10 node network size is shown not to be significant ($P \sim= 0.12$).

7.5.3 Conclusions

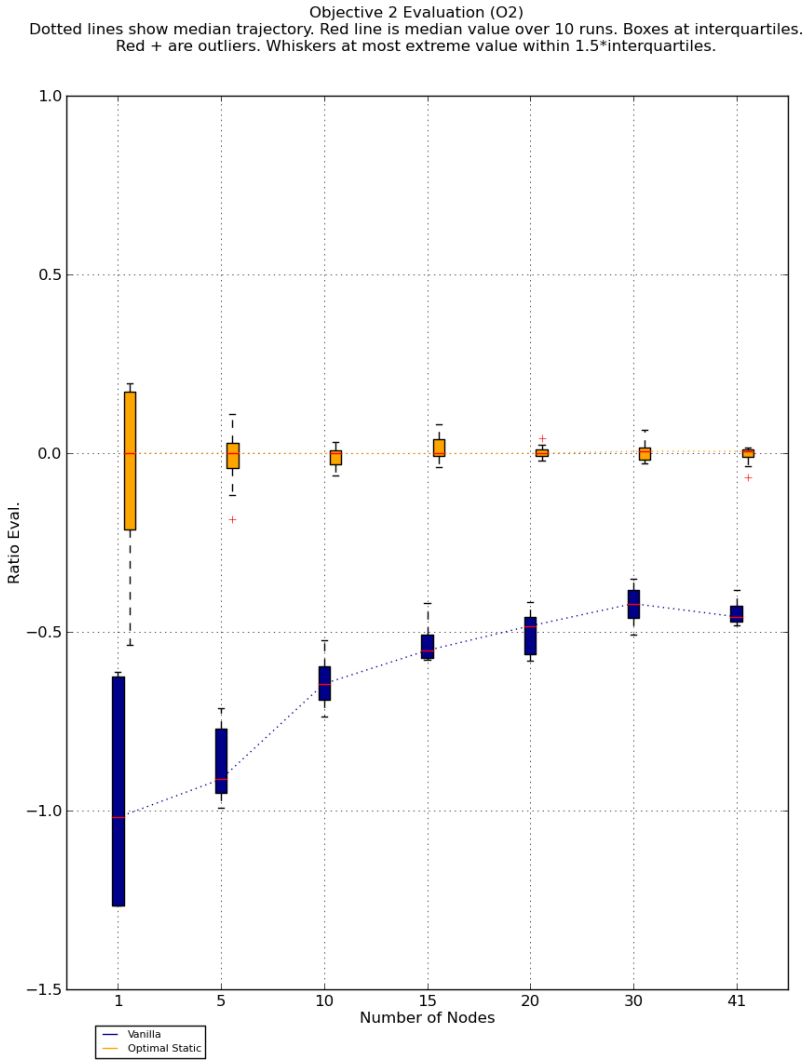
In this study we retested our *CARDINAL-Vanilla* model implementation on a 41 node enterprise network using the experiment design, procedure, parameter configuration and evaluation approach cloned from our previous virtualised tests.

Surprisingly, we found far more stability from CARDINAL-Vanilla on the detector transmission quantity measure (*Dist-M1*) compared to its virtualised tests results, and significant correlation between the two algorithms. However, corroborating earlier findings from the virtualised tests, the engineered algorithm (Optimal-Static) reported significantly better suitability on larger-sized networks (>15 nodes) than the CARDINAL algorithm with respect to its system performance for SCADA-like networks (O2) and for enterprise networks (O3).

In this real use case scenario experiment we find both algorithms show *Dist-M1* and *Time-M2* scores similar to one another. The important discrepancy between the two algorithms is in the *DataSent-M3* metric. One can argue that Optimal-Static's algorithmic approach is impossible to recreate as it has no need to request or receive the model state of the other nodes, it assumes perfect behaviour of the nodes – i.e. no loss of data in transmission, no disconnects, no detectors deleted or corrupted etc. Thus a true and realistic comparison of the two methods remains as open work and will incur extra transmissions and transmitted data.



(a) Metric observations. Number of detectors distributed during experiment.



(b) Immunisation rate suitability upon SCADA-like networks (O2), combining metric M1,M2,M3 data using our extension of the Weisbin & Rodriguez ratio equation with OptimalStatic as reference. Note that plot y-axis is truncated to -1.5 excluding Optimal-Static's outlier value transposed from the *Time-M2* metric results.

Figure 7.9 – Box plot observations of detector distribution quantity metric and multi-objective evaluation (O2) on the enterprise network tests.

While these amounts are likely to be less than the roughly doubled data quantity differences there do remain adjustments to consider in order to lower the bio-inspired algorithm's transmission quantities. Within the constraints of no external model knowledge, the single most expensive data transmission are imposed by redundancy in the peer-to-peer bidirectional graph mapping that connect the nodes. Secondly is the redundancy of retransmitting detectors or transmitting unimportant detectors. *CARDINAL-Vanilla*'s presented heuristic measures importance based on commonality, whether commonality actually carries the opposite definition, i.e. of unimportance, is a point of discussion. An alternative might be of multiple metrics contributing to a single definition of importance or separated definitions for each individual neighbouring node.

7.5.4 Next Steps

The next item of work, we will look deeper at the differences between the virtual and enterprise network test set-ups and the discrepancies in their results. We believe the culprit difference between the two experiments is the delay and ordering of execution, including I/O, affecting both network transmissions and dataset file read-in. These are the items that we will investigate to understand those differences.

7.5.5 Future Work

We know that data transmission (*DataSent-M3*) was the primary measure that led to CARDINAL's poorer performance evaluation. The cause for the additional data was the retransmissions of already sent detectors; this action led to additional time spent in selection, sending, receiving and processing of the received detectors. Investigation of a hybridised and more intelligent approaches are recommended for future work for the priority heuristic definition, detector and destination selection mechanisms and the transmission quantity decision used to respond to monitored danger or with the requirement of greater intensity.

7.6 Further Analysis of Network Testing

7.6.1 Objectives

The major challenge of evaluating software intended for a real environment is how to interpret and translate its performance in modelled, simulated or virtualised environments, with respect to the real world behaviour of that system. In comparison to the Optimal-Static algorithm, CARDINAL-Vanilla's results in the real network tests were relatively stronger than the poorer performance in the virtual network tests. This further analysis work aims to understand the cause of those differences.

The two earlier comparative benchmark studies showed discernible differences. For example the *Time-M2* median scores for the CARDINAL-Vanilla algorithm were 308.47 seconds in the virtual network with 20 nodes and only 23.622 seconds in the enterprise network tests with 41 nodes. The *Distributed-M1* medians scores for the quantity of detectors distributed were 10 and 14 for the same tests, representing 50% and 70% of the required detectors distributed.

An hypothesis is that the *Time-M2* differences are caused by the number of cores executing the tests, 2 cores verses 82 cores. However, the *Distributed-M1* difference is not straightforward. The discrepancy may have been caused by execution differences in the two test conditions or perhaps the time desynchronisation used in the virtual network tests or the extended time taken during the virtual tests thus affecting the internal model state of the CARDINAL algorithm. Further more, other forms of uncontrolled noise, as stated in 6.2.5, may have caused an effect. Of particular likelihood to cause these observed effects is the operating system-specific process execution behaviour.

This study will investigate the comparative effects of network virtualisation on execution behaviour (7.6.2), the effects of virtual network time desynchronisation in terms of metric differences (7.6.4) and in terms of detector delivery at specific nodes (7.6.5).

7.6.2 Observations of Time Synch/ Desynch in Network Virtualisation

The following is analysis of execution behaviour between three forms of time synchronisation in virtual network tests and the earlier enterprise network tests. The following four configurations are the target of this investigation to draw observational comparisons of execution behaviours:

- **(a)** shows virtual network environment with time synchronised execution (0 s delay).
- **(b)** shows virtual network test with 0.45 sec between node start times, matching the earlier virtual network tests (in 7.2).
- **(c)** shows virtual network test with 0.9 s between node start times.
- **(d)** shows enterprise network test with time synchronised execution (0 s delay), matching the earlier enterprise network tests (in 7.6.3).

Figure 7.10 shows a single exemplar execution of the benchmark CARDINAL-Vanilla algorithm through each experiment phase event under the four different test conditions. These states include start and finish of training and testing phases and the other states specified in 6.3. The diagram follows the conventional sequential diagram format, however it is downscaled for presentation purposes. Sequential execution time runs from top to bottom. Each vertical column within each sequence diagram is an architecture client process. Each column represents the execution of a process's experiment phases. In Figures 7.10 **(a), (b)** and **(c)** all processes execute on a single computer. In Fig. 7.10 **(d)** each process executes on a different computer.

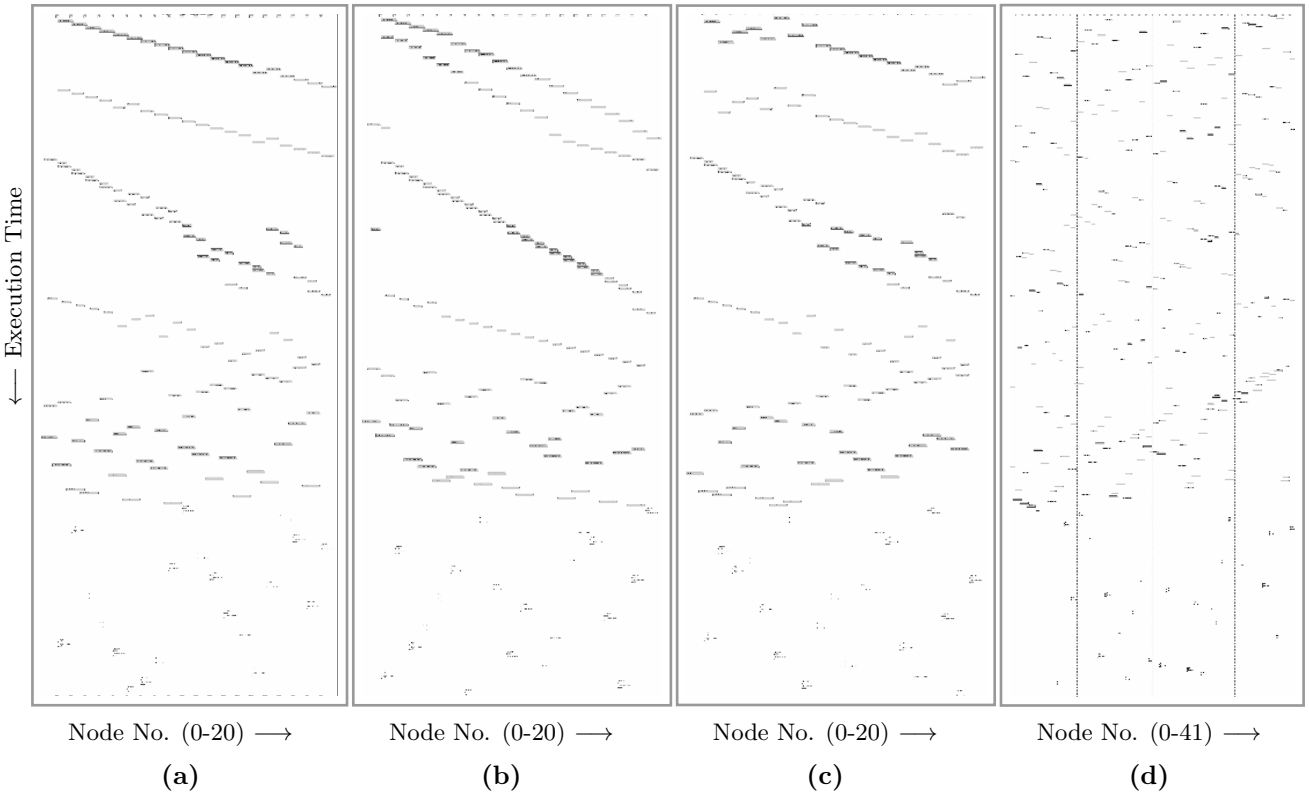


Figure 7.10 – Sequence diagrams showing execution of experiment phase events w.r.t time. Time and ordered node numbers are plotted on Y-axis and X-axis. Black blocks show the Experiment Phases (including start and finish states) stated in 6.3 running on each node. *Note the vertical-dashed lines on* **(d)** *are an image scaling effect.*

The maximum tested number of nodes per trial has been selected to express the test conditions under maximum tested load. The decision to include the enterprise network test **(d)** sequence behaviour is in order to illustrate that the experiment phase event transitions are independent at each node, except at the *Shutdown* phase event typically at 60% time. This virtualised vs non-virtualised difference is observable by the contrasting execution behavioural differences. In Figure 7.10, due to the difference in the numbers of nodes, neither correlations nor comparisons from **(d)** to other virtual tests should be concluded.

7.6.2.1 Observations

In Figure 7.10, we can make informal observations of the variation in execution dependency with respect to starting order and duration of CPU time allotted per process per execution time slot. Table 7.9 shows a summary of these observations.

The informal *Dependency upon Other Processes* ($Obsv1$) measure is separated into *high, med, low* and *none* categories and the informal *Allotted CPU Time per Process per CPU Time Slot* ($Obsv2$) measure has *small, med, large* and *unknown* categories. The definition of each category is given with context in the observation descriptions below.

In the time synchronised virtual test **(a)**, there is a visually obvious execution dependency on the node number order (from left to right) and on the completion of the *Start-Up* and *Start Training* phases; as the top two diagonal lines in Fig **(a)** exemplify. This indicates a 'high' category of the $Obsv1$ measure. The *Shutdown* phase events are lower in each diagram – this phase is dependent on the states of other nodes and as such are not executed in node order. In **(a)** there is a tendency that the first process waits until all processes have completed a phase before beginning a following phase. This indicates a 'small' category of the $Obsv2$ measure.

In the 0.45s time delayed virtual test **(b)** shows some differences from **(a)**. The first node reaches most experiment phases earlier than other nodes, exemplified by the three top diagonal lines in **(b)**. This indicates a 'high' category of $Obsv1$. Secondly, there is a tendency toward a process completing more experiment phases sequentially in time; this suggests the allotted CPU execution time is greater per process than **(a)**. This indicates a 'medium' category of $Obsv2$.

In the doubled time delayed virtual test **(c)**, there is a some visual evidence of execution dependency on the node number order (from left to right). In fact, node six is first to execute in the early phases. This observation is similar to, but less distinguished than, the top two diagonal lines in **(a)** and **(b)**. This indicates a 'medium' category of $Obsv1$. Similarly to **(a)**, in **(c)** less phases are completed per time slot than in **(b)** indicating a 'small' category of $Obsv2$.

In the enterprise network test **(d)** that same dependency between processes is not noticeable in the early experiment phases. This indicates a 'none' category of $Obsv1$. This differs, as expected, at the *Shutdown* phase (at 60% time) which is dependent upon all nodes completing the experiment phases for *Training* and *Testing*. In **(d)** CPU time execution per node is dedicated to, not shared by, a single node and thus CPU time share is unknown, indicating an 'unknown' category of $Obsv2$.

	No Delay (a)	**0.45s Delay (b)**	**0.9s Delay (c)**	**Enterprise (d)**
(*Obsv*1) *Dependency upon Other Processes*	high	high	med	none
(*Obsv*2) *Allotted CPU Time per Process per CPU Time Slot*	small	med	small	unknown

Table 7.9 – An informal summary of sequence analysis results from informal observation of an exemplar run of CARDINAL-Vanilla under different conditions: (a), (b), (c) and (d). Observed categorical values are given in the ranges *(high,med,low,none)* and *(small,med,large,unknown)*.

The effect of the different environments on median total execution time of an experiment run, over 10 runs, is 317.1 seconds with a sampled standard deviation (Std) of 8.1 to 1 significant digit in **(a)**, is 325.6 seconds (Std = 5.2) in **(b)** and where we double the time delay to 900 ms is 327.9 seconds (Std = 9.5) in **(c)**. For the same activity this duration is 39.4 seconds (Std = 2.1) in **(d)**.

	No Delay (a)	**0.45s Delay (b)**	**0.9s Delay (c)**	**Enterprise (d)**
Experiment Runtime η	317.1	325.6	327.9	39.4
Experiment Runtime (Std)	8.1	5.2	9.5	2.1

Table 7.10 – Table of total experiment runtimes measured in seconds of CARDINAL-Vanilla under different conditions: (a), (b), (c) and (d). η is *(Std)* are medians and standard deviations over 10 trial runs.

7.6.2.2 Discussion

This informal sequence execution analysis has presented observational differences in CPU execution time and levels of dependency behaviour between virtual and enterprise network tests and dependent upon each virtual test time synchronisation condition. This analysis is relevant to illustrate the execution behaviour differences between the time synchronisation delays under experimental test conditions. Due to the informality of the measures and the variation over alternative CPU architectures and alternative operating system process scheduling algorithms only observations, not conclusions, should be drawn from this analysis.

7.6.3 Effects of Time Desynchronisation

The time desynchronisation used in the virtual tests and described in 7.2.4.1 is a possible reason to have caused the differences in the *Distributed-M1* results. Note that the enterprise tests were synchronised.

The next two sections will describe the effects of time desynchronisation upon metric results and upon detector arrival times. The hypothesis 'the metric difference in *Distributed-M1* between the virtual network tests and enterprise network tests is caused by the time desynchronisation value' is evaluated in section 7.6.4. The hypothesis 'the detector arrival time difference between the virtual network tests and enterprise network tests is caused by the time desynchronisation value' is evaluated in section 7.6.5. In both tests the virtual network trials, there is one variable, the desynchronisation time delay value of 0 seconds, 0.45 seconds and 0.9 seconds.

7.6.4 Metric Differences with Time Desynchronisation

The following is an analysis of the measured differences between only the virtual tests when that desynchronisation delay is changed. We conclude with metric differences between the enterprise network tests and either using synchronised or desynchronised virtual network testing. The hypothesis for this section is that 'the metric difference in *Distributed-M1* between the virtual network tests and enterprise network tests is caused by the time desynchronisation value'.

Here we re-ran the virtual tests with the benchmark CARDINAL-Vanilla algorithm, over 20 runs, with only a change in the starting time delay issued between each node. Section 7.2.4.1 described the use of the 0.45 second value between the starting of each nodes during the benchmark tests.

	0.45s Delay		**0.9s Delay**		**No Delay**		**Kruskal-Wallis Test**		
n	η	**(Std)**	η	**(Std)**	η	**(Std)**	H	df	*P-value*
				Distributed-M1					
1	11.0	(0.0)	11.0	(0.0)	11.0	(0.0)	Nan	19	N/A
5	12.0	(0.768)	12.0	(0.872)	12.5	(0.887)	2.722782	19	0.256304
10	13.0	(0.812)	13.0	(0.698)	13.0	(0.748)	4.342295	19	0.114047
15	12.0	(1.276)	11.0	(1.64)	11.0	(1.424)	2.729407	19	0.255456
20	11.0	(1.411)	10.0	(1.83)	11.0	(1.114)	4.768778	19	0.092145
				Time-M2					
1	0.669	(0.694)	0.253	(0.745)	0.318	(0.404)	3.515293	19	0.172450
5	30.23	(3.22)	28.569	(3.34)	28.144	(3.057)	2.999344	19	0.223203
10	87.585	(5.139)	79.161	(9.798)	83.349	(7.172)	11.899016	19	**0.002607**
15	167.144	(9.02)	142.087	(12.739)	163.409	(5.296)	30.779344	19	$\mathbf{2.071812}E^{-7}$
20	263.056	(7.754)	258.008	(30.892)	254.56	(7.96)	13.526885	19	**0.001155**
				DataSent-M3					
1	0.5	(0.068)	0.6	(0.065)	0.6	(0.074)	5.426206	19	0.066331
5	11.35	(0.547)	11.65	(0.338)	11.7	(0.59)	6.753965	19	**0.034150**
10	57.7	(1.231)	54.6	(1.219)	55.05	(1.546)	30.781278	19	$\mathbf{2.07}E^{-7}$
15	130.9	(1.996)	120.0	(1.868)	121.4	(1.401)	39.852917	19	$\mathbf{2.218447}E^{-9}$
20	241.45	(3.448)	216.45	(3.241)	215.85	(1.84)	39.488576	19	$\mathbf{2.661734}E^{-9}$

Table 7.11 – Summary of metric results of virtual network tests with varying starting time delays between nodes. Table shows 5 network sizes (n) and results of metrics *Distributed-M1*,*Time-M2*,*DataSent-M3* data over 20 iterations. η and *(Std)* show medians and sample standard deviations. The Kruskal-Wallis null hypothesis shows whether the mean ranks of the 3 tests are the same, critical $\alpha = 0.05$.

7.6.4.1 Results

In Table 7.11 we report the effects of removing that delay (No Delay), a repeat of the benchmark test (0.45s Delay) and doubling that delay (0.9s Delay). Table 7.11 includes a Kruskal-Wallis Test (KWT) (Kruskal & Wallis, 1952) variance analysis of the three sets of test results, showing whether the mean ranks of the tests samples are the same. Where P is less than the critical $\alpha = 0.05$ the null hypothesis of mean rank similarity is rejected, meaning at least one sample dominates another sample and an implication that their mean ranks are different.

The significant KWT results in Table 7.11 show mean rank difference in *Time-M2* and *DataSent-M3* results at $n = 10, 15, 20$, i.e. as the number of network nodes (n) increases these metric results differ. With respect to *Distributed-M1*, the KWT results are not significant at any n size. Therefore the mean rank is similar over all trial time delays and over all network sizes (n) with respect to *Distributed-M1*.

Furthermore, the Mann-Whitney U tests in Table 7.12 with corrected two-tailed P-value[1], show population distinction over No Delay and 0.45s Delay *Distributed-M1* results. To assess significance we have corrected the critical α to 0.0167 with $\alpha/3$, to adjust for 3 additional tests upon this sample data (Field & Hole, 2003)[p247]. Where the $P < \alpha$, we reject the null hypothesis of difference due to random sampling and conclude instead that the samples are of different populations. In Table 7.12, each network size test for difference shows a P-value of $> \alpha$ suggesting that the No Delay and 0.45s Delay *Distributed-M1* results are similar. The P-value at $n = 20$ gives particularly compelling evidence of this.

Mann Whitney U Test: *Distributed-M1*			
n	*U*	*df*	*P-value*
5	147.5	19	0.132746
10	140.0	19	0.08642
15	140.5	19	0.101054
20	166.5	19	0.35674

Table 7.12 – Mann-Whitney U two-tailed test upon No Delay and 0.45s Delay *Distributed-M1* results from CARDINAL-Vanilla, with adjusted critical α as 0.0167. Showing metric samples of a similar population.

We follow with two additional Mann Whitney U tests for difference. In Table 7.13a we compare 0.45s Delay *Distributed-M1* results with the enterprise network results of CARDINAL-Vanilla, which as we have already seen are distinctly different in Figure 7.5a and Figure 7.9a. The test P-values corroborate this. Above $n = 5$ all P-values are less than $\alpha = 0.0167$, thus implicating different population origins. In Table 7.13b we look for a measure of statistical difference in *Distributed-M1* results between No Delay and the enterprise CARDINAL-Vanilla tests and find greater likelihood of same metric behaviour at $n = 5$ than in the 0.45s Delay at $P = 0.362$ verse $P = 0.108$. However, the P-values in the remaining tests indicate results of distinctly different population.

[1]Python's statistical SciPy library implementation of the Mann Whitney U performs a one-tailed test. This can be converted to two-tailed with $P \times 2$ according to the SciPy test documentation.

Mann Whitney U Test: *Distributed-M1*			
n	*U*	*df*	*P-value*
5	30.0	9	0.108518
10	15.0	9	**0.001826**
15	5.0	9	**0.000222**
20	5.0	9	**0.000222**
*30	8.0	9	**0.000967**
*41	7.0	9	**0.000719**

(a)

Mann Whitney U Test: *Distributed-M1*			
n	*U*	*df*	*P-value*
5	38.5	9	0.362283
10	5.0	9	**0.000189**
15	0.0	9	$\mathbf{5.8}e-05$
20	0.0	9	$\mathbf{5.3}e-05$
*30	0.0	9	$\mathbf{9.4}e-05$
*41	0.0	9	$\mathbf{9.3}e-05$

(b)

Table 7.13 – Mann-Whitney U corrected two-tailed test for population difference of CARDINAL-Vanilla over 10 runs, comparing *Distributed-M1* results, with adjusted critical α as 0.0167. Sizes *30 and *41 compare the virtual *Distributed-M1* results at n=20 with the enterprise *Distributed-M1* results at sizes $n = 30$ and $n = 41$.
(a) Shows results of 0.45s Delay and enterprise network tests.
(b) Shows results of No Delay and enterprise network tests.

7.6.4.2 Discussion

This analysis has shown the effects of time desynchronisation on the virtual network tests using the CARDINAL-Vanilla algorithm.

The Kruskal-Wallis tests have shown no significant difference between the time delays with respect to the *Distributed-M1* metric results (P_1) within our tested time delay variations. It has reported significant differences in the *Time-M2* in network sizes ≥10 (P_2) and differences in *DataSent-M3* in network sizes ≥5 (P_3). While $P_2 \wedge P_3 \implies Q$ difference between virtual and enterprise results, $P_1 \implies \neg Q$; therefore P_1 denies the outright difference. We proceeded by further investigating the *Distributed-M1* result differences.

The first Mann Whitney U test upon the *Distributed-M1* (detectors distributed) results has shown compelling similarity, no significant difference, between the No Delay and 0.45s Delay runs. The latter Mann Whitney U tests upon *Distributed-M1* results show significant difference between both No Delay and 0.45s Delay runs against the enterprise test results in network sizes ≥5. This indicates that time desynchronisation is not the cause for *Dist-M1* metric result discrepancy between the virtual and enterprise network test results presented earlier in the chapter. $\neg P_1 \wedge P_2 \wedge P_3 \implies Q$ gives the logical consequence of difference but is not valid w.r.t the metric results. Therefore we reject the first hypothesis due to the insignificant difference between the *Dist-M1* metric results with the changed delay value.

7.6.4.3 Conclusion

Based on these results, we reject the first hypothesis due to the insignificant difference between the *Dist-M1* metric results with the changed delay value. We would expect that the *Distributed-M1* results in tests with synchronised start times will not be significantly different from the benchmark results presented earlier. However, we would expect result differences in *Time-M2* at network sizes ≥ 10 and *DataSent-M3* at network sizes ≥ 5.

7.6.5 Detector Delivery Time Differences with Time Desynchronisation

The following is an analysis of testing the hypothesis that the time delay used in the virtual tests has an effect upon detector delivery times under the tested configurations.

Within the benchmark CARDINAL-Vanilla algorithm, we expect detector delivery at a destination node to be approximately randomly ordered with respect to node starting indices. This is due to the random component of the transmission destination node selection mechanism (stated in 5.10.2). The SendAll and OptimalStatic algorithms (see 7.2.2) destination selectors will choose all known connected nodes in order, therefore the detector arrival times should be approximately linearly ordered with respect to the node indices. The time delay under investigation also affects nodes in order. If the time delay does affect detector receipt times then a correlation between node starting order and detector delivery times is expected. Therefore, this analysis will test for this correlation under each test configuration.

7.6.5.1 Results

The first network transmitted arrival time at each node is collected from the logs of a detector signature from Dataset D (6.7.1). The detector chosen had the highest frequency in the training set to ensure the greatest likelihood of transmission to all nodes using all of the benchmark algorithms, and specifically including the CARDINAL-Vanilla algorithm's detector transmission selector stated in section 5.10.3. Explicitly, this is not the time it was read from the dataset as the read-in is not guaranteed at only one location since multiple nodes may read-in and transmit the same detector. This collection is repeated for each trial and each test configuration as follows.

7.6.5.2 Effect upon Virtual Benchmark Tests

Figure 7.11 reports a clear order to the arrival times at each node. Here the results are shown from the benchmark (0.45s Delay) virtual network tests for algorithms CARDINAL-Vanilla, OptimalStatic and SendAll. Table 7.14a reports the Spearman's rank *rho* test for correlation of the median arrival time values to the node starting index order values. Across each algorithm under this configuration, significant correlation is implied by P-values below $\alpha = 0.05$. These tests use Spearman's rank two tailed test on without correcting for value tie cases (Spearman, 1904). Kendall's Tau is the alternative that would correct tied values with an additive affect upon the P-value, which we choose not to use as we accept that nodes may share the same median arrival times.

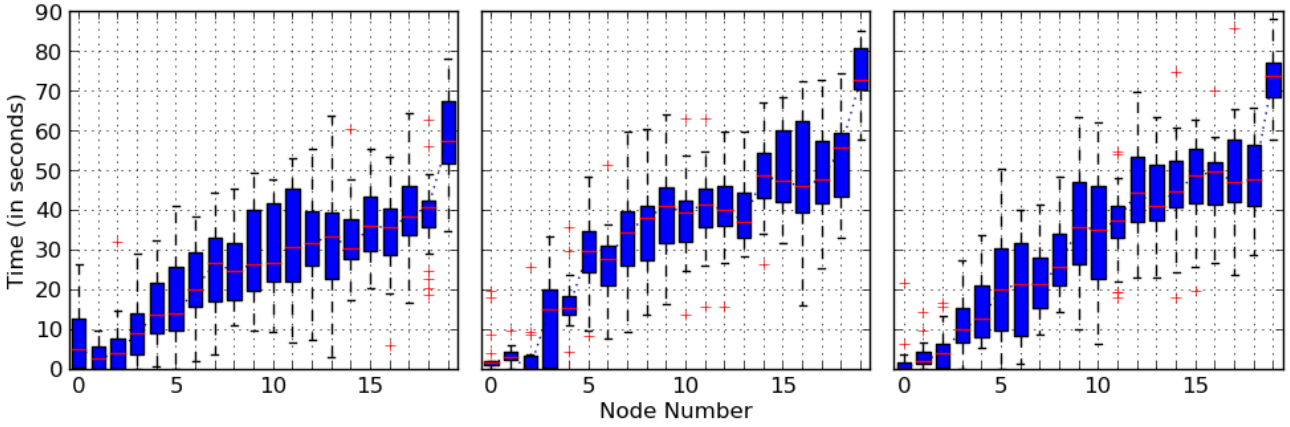


Figure 7.11 – Virtual network benchmark tests with 20 (0-19) nodes over 25 runs. 450ms time delay between each node start time. Plots from left to right show findings from algorithms: CARDINAL-Vanilla, Optimal-Static, Send-All.

7.6.5.3 Effect upon Enterprise Benchmark Tests

By contrast in Figure 7.12 the enterprise tests without the time delay over algorithms CARDINAL-Vanilla and OptimalStatic visually show arrival times in a random-like order. Table 7.14b reports the Spearman's *rho* rank correlations as insignificant in both tests scenarios. A P-value at 0.65 for CARDINAL-Vanilla implies sampling of random order. Less insignificantly at $P = 0.15$ for the Optimal-Static algorithm, because its selection algorithm chooses the transmission destinations in natural order.

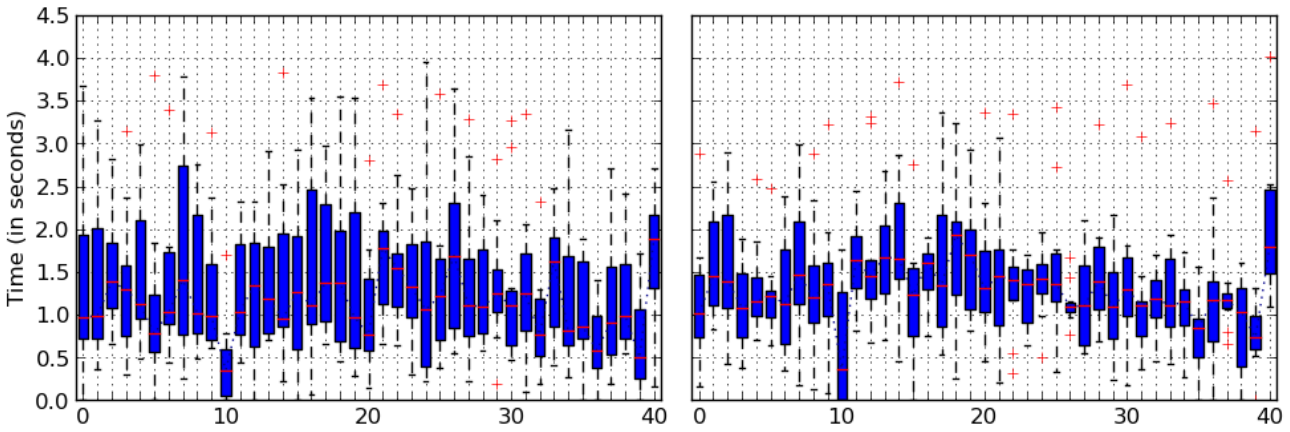


Figure 7.12 – Enterprise network benchmark tests with 41 nodes (0-40) over 10 runs. No time delay between node start times. Plots from left to right show findings from algorithms: CARDINAL-Vanilla, Optimal-Static.

7.6.5.4 Effect upon Virtual Time Delay Tests

Next we look at the effects of time delay changes of 0 seconds, 0.45 s and 0.9 s in the virtual network tests, at network size 20. Figure 7.13 shows the detector delivery time behaviour over 20 runs. The central plot shows the repeated test runs of 0.45s Delay CARDINAL-Vanilla which should be very similar to left plot in Figure 7.11. We observe similar behaviour between No Delay and 0.9s Delay arrival times.

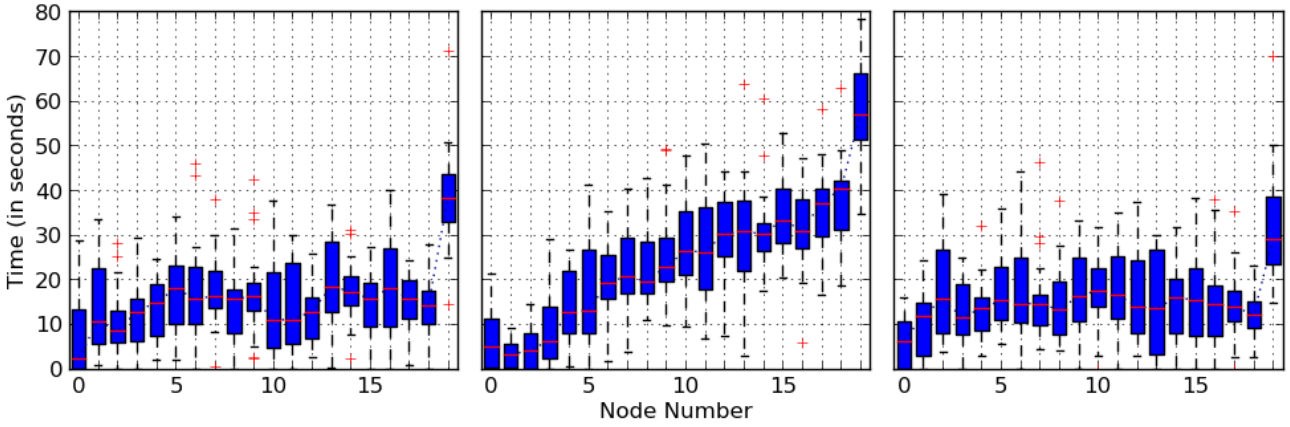


Figure 7.13 – Plots show time in seconds (Y-axis) for the most common detector to be received via network at each numbered node (X-axis) (in starting order). Time is relative from earliest receipt time. Virtual network benchmark tests with 20 (0-19) nodes over 20 runs running CARDINAL-Vanilla. Plots from left to right show findings from time delays: None, 0.45 s and 0.9 s.

Table 7.14c reports significant correlation at $P < 0.01$ in the No Delay and 0.45s Delay tests, where critical α remains at 0.05. This indirectly indicates that the synchronised virtual test (No Delay) share significance with respect to correlation between detector delivery timestamps to node starting order, at a network size of 20. Explicitly, the No Delay result of $P = 0.011$ shows less correlation than 0.45s Delay's with $P = 1.37E^{-15}$ to node index order. Removing the time delay reduced the distinctiveness of detector delivery order – i.e. the order is more randomly ordered. However, both are significantly correlated to the ordered effect. The 0.9s Delay tests fails to suggest a correlation.

	Vanilla	Optimal-Static	SendAll
rho	0.980451	0.953383	0.984962
P	$\mathbf{3.68936}E^{-14}$	$\mathbf{8.33305}E^{-11}$	$\mathbf{3.53643}E^{-15}$
df	24	24	24

(a)

	Vanilla	Optimal-Static
rho	−0.0729158	−0.224216
P	0.65051	0.158746
df	9	9

(b)

	No Time Delay	Time Delay (0.45s)	Time Delay (0.9s)
rho	0.553383	0.986466	0.374436
P	**0.0113704**	$\mathbf{1.37757}E^{-15}$	0.103841
df	19	19	19

(c)

Table 7.14 – Tables show Spearman's (*rho*) rank correlation statistic and significance (*p* is the P-value) under different test conditions. Each test attempts to correlate ranked detector delivery times via network transmission per node with starting node indices. Critical $\alpha = 0.05$.
(a) Table shows benchmark virtual network tests over 25 runs, 20 nodes.
(b) Table shows benchmark enterprise network tests over 10 runs, 41 nodes.
(c) Table shows time delay virtual network tests over 20 Runs, 20 nodes.

7.6.6 Discussion

This work has shown that time desynchronisation in the virtualised network tests has had a small effect upon the sequence of detector delivery times; however, its effect is shown to be insignificant. Testing in our virtualised network with No Delay and 0.45s Delay have both shown a significant and undesirable correlation between detector delivery and node index. If we increase the desynchronisation to 0.9s the correlation effect is no longer significant, however this then has a degraded *Distributed-M1* (distributed detector quantity) result, as 7.6.4 has previously shown.

In the enterprise network tests the detector delivery behaviour is as expected according to the algorithm descriptions. Neither CARDINAL-Vanilla nor Optimal-Static show significant correlation to node indices; however, Optimal-Static is more closely statistically rank-correlated to node index order. The boxplots and rank correlations indicate no significance is to be found, therefore we have not completed a test for correlation between the detector delivery times in the virtualised network tests and in the enterprise network tests.

The Spearman's Rank Rho tests for correlations of detector arrival times to node index order in Table 7.14c for 0 second and 0.45 second time delay are significant under the virtual network environment (P_1). The Spearman's Rank Rho tests on the enterprise network with no time delay in Table 7.14b are each insignificant (P_2). $P_1 \neq P_2 \implies \neg Q$, therefore we reject the second hypothesis. This analysis determines that the detector arrival time difference between the virtual network tests and enterprise network tests is not caused by the time desynchronisation value.

7.6.7 Conclusion

The summary of this further analysis is that the virtual network execution behaviour had an, as yet, unresolvable difference to real network testing. Both hypotheses of virtual to non-virtual difference effect as caused by the time desynchronisation delay have been rejected. The resounding viewpoint is that interpreting the results of the real network tests removes ambiguity. Thus if hard supporting evidence for future tests of the architecture is needed, then those tests should be run on real networks under the similar constraints as the intended target conditions.

This further analysis upon the benchmark network testing has investigated the effects of network virtualisation and time desynchronisation. This has been done with the intent of recovering the difference between the real and virtual benchmarked network testing types. The effects of network virtualisation section has visualised and observed execution behavioural differences between tests, including different delays under the virtual network and in the enterprise network. The metric differences with time desynchronisation work has shown that synchronising and desynchronising by 0.45 seconds in the virtualised network tests has had a small effect upon the sequence of detector delivery times; however, it's effect is shown to be insignificant. We can find a distinct difference between delays in the virtual tests, but none of our tested delay changes have caused the virtual and enterprise tests to report equal or equally distributed results for metric *Distributed-M1*. The detector delivery differences work has shown that time desynchronisation in the virtualised network tests has had an insignificant effect upon the sequence of detector delivery times. Tests with No Delay and 0.45s Delay have both shown a significant and undesirable correlation between detector delivery and node index. Increasing the delay reduced the correlation but also degraded the *Distributed-M1* metric performance result. The further analysis work here did not find the cause of the difference between the *Distributed-M1* results nor has it found a solution to cause the virtual network tests to behave similarly to the enterprise network tests. Even synchronising the start times in future virtual network tests will give *Distributed-M1* performance results different from the enterprise network tests.

7.6.8 Future Work

The virtual network results presented are collected from tests run on a PC with two cores. We did try running an earlier version of the software with 96 cores of 1165 Mhz on a single machine but found that *Time-M2* scores were slower than the two core execution for some small tests. This work suggests that more cores is likely to improve the time for detector delivery (*Time-M2*) metric result and may have a positive effect upon the *Distributed-M1* results. Further work in this direction may be rewarding. Alternatively replacing the time dependent decision components in CARDINAL and CARDINAL-Vanilla may improve *Distributed-M1* performance measures.

7.7 Chapter Conclusion

These studies achieved the first distributed implementation-level benchmark results of *CARDINAL-Vanilla*, a *fair* implementation of the *CARDINAL* model by (Kim *et al.* , 2005), and found poorer performance by the AIS approach than an equivalent engineered approach.

This chapter gave observations with which to assess the key hypothesis set in section 1.5. The key statistical tests results for the virtual network trials can be found in Table 7.4 on page 122, whereas the enterprise network trials can be found in Table 7.8 on page 136. Based on the *CARDINAL-Vanilla* architecture, conducted in the stated methods and parameter configurations, the following conclusions are drawn.

For low-throughput networks, such as industrial control networks, as measured by the objective O2, the findings indicate that the hypothesis is rejected in all network size cases under virtual and enterprise network trials.

For high-throughput networks, such as enterprise networks, as measured by the objective O3, the findings indicate that the hypothesis is rejected in all network size cases under virtual and enterprise network trials, except at network size 10 where the AIS algorithm performs insignificantly better than the engineered approach.

The amount by which the results are poorer is not relevant to a real life application. The enterprise network results in section 7.4 showed comparable *Distributed-M1* and *Time-M2* metric performances to the engineered selection and distribution algorithm, however more data was sent by CARDINAL-Vanilla and led to an overall poorer immunisation rate performance. The virtual network results in 7.2 reported poorer *Distributed-M1* distribution performance as the network size increased by comparison to the real networked tests. Our further analysis in 7.6 was unable to resolve that difference and as such we concluded that the likely cause is related to the reaching of capacity bottlenecks of the virtual networks on a single workstation. The resounding viewpoint that the further analysis gave was that the real network tests removed ambiguity from the interpretation of the results.

7.7.1 Next Steps

The real networked experiment showed the important discrepancy between the engineered and AIS algorithms was the *DataSent-M3* metric. Optimal-Static is described in section 7.2.2, we argue that its algorithmic approach is impossible to recreate as it has no need to request or receive the model state and is thus prone to very poor performance under failure. Therefore adding further data transmissions and further data sent would harden its approach. Thus, it is possible that improving CARDINAL-Vanilla's parameter configuration could lead to better performance. Additionally, reducing the quantity of open communication sessions required by a peer-to-peer connectivity graph could lead to similar performance robustness while reducing the data transmissions. The first of these two items is contained in our next chapter of work.

7.7.2 Future Work

In section 7.4 we recognised that *CARDINAL*-Vanilla's use of commonality as a measure of priority is an avenue for further enquiry leading to better performance. In particular the task is to find a combination of multiple measures of importance, this may include input rarity and commonality. Additionally we

found that the probability of a detector's selection is dependent upon the order and repetition of inputs within each dataset. By adding an adaptive mechanism based on incoming data the *under-attack* condition and detector selection mechanism are likely to perform better.

The priority heuristic approach that CARDINAL-Vanilla offers leads to a predictive assessment of *what-content-is-needed-where* without requesting any state knowledge. In peer-to-peer networks a single update request for the full network leads to either a naïve near exponential (n^{n-1}), or key lookup-based $O(Log(n))$ (Rowstron & Druschel, 2001),(Stoica *et al.* , 2001) quantity of transmissions, which becomes a big problem as the network size increases and the update interval reduces. With a more intelligent evaluation of the core principle of distribution, possibly probabilistic using alternative metrics or related to (Censor-Hillel & Shachnai, 2011), a hybrid of CARDINAL-Vanilla's system could be a feasible solution.

Chapter 8

Parameter Tuning

8.1 Introduction

In this chapter we will investigate the behavioural differences of each architecture parameter upon our global metrics. Our interest is in discovering which parameter configuration is best for our Industrial Control System (ICS) and Supervisory Control and Data Acquisition (SCADA) network application destination. Secondarily, how does each parameter affect the metric performance; by answering this we can adapt the architecture parameters to suit the runtime condition of any operational network or workstation.

In section 8.2 we will begin by evaluating each parameter's value range. The set of key architecture parameters were selected in section 5.12 and defined with a continuous range and a discrete set of values per parameter. The first evaluation will show only independent parameter performance in the localised parameter space, we will follow up with a gradient ascent search to more thoroughly evaluate the parameter space on our virtual network experiment testbed in 8.4 and on our enterprise network in 8.6.

8.2 Range Testing on Virtual Networks

8.2.1 Objectives

In the parameter range test series we want to understand how each parameter affects the metric performance, such that we can set, or in future adapt, the architecture parameter configuration to suit our use case scenario. This leads us to three questions. First, is there a correlation between the parameter value and a metric result, and if so is the result affected in a monotonic order. Secondly, is the effect one parameter has on a metric result larger than the effect of another parameter on the same metric. Thirdly we ask which parameter set-up gives us the best immunisation rate performance. The latter question carries the qualifier, as measured from the local point in parameter space.

We can answer the first by checking whether any of the metric results are monotonically correlated to the ordered parameter value tests. The second question can be answered by checking the magnitude of the metric differences between each parameter value's change, ideally where the range value intervals are equivalent. The third can be answered with a comparison of our multi-objective evaluation systems for immunisation rates, which are described in 6.9.

First we will describe the experiment test set-up and discuss our metrics in these tests, then show the results of our analysis.

8.2.2 Experiment Design

This experiment followed the same design set out in the virtual network benchmark tests 7.2.4. The differences are the dataset and the range of parameter variables under test which we have described in Table 8.1. We have fixed the quantity of network nodes to five with 10 repeated trials and the time desynchronisation is set at 0.45 seconds, identically to 7.2.4 and as analysed in 7.6.3.

The execution script was adapted with a for-loop to wrap over a set of input configuration files, where each file contained a varying set of the parameter configuration values. The CSIC Dataset C was used, see 6.7.1.

8.2.2.1 Parameter Testing Ranges

Each parameter has been discussed with respect to the functions and agents that it affects within the architecture in chapter 5. Within 5.12 we described each parameter and its range, these have been copied into Table 8.1 for ease of reading.

As a recap, the $P0$ parameter is the Under Attack Volume Percentage, which determines the size and destination quantity of transmissions made while an architecture node is in the under attack condition. $P1$ is the Initial Priority Value Multiplier, which gives an initial priority value to newly created detector/response modules, thus affecting its likelihood for transmission. $P2$ is the Priority Value Suppression, which reduces the priority of rarely needed modules. $P3$ is the Static Moving Window Size, which defines the duration of time over which *recency* can be considered; it affects the thresholds within the under attack decision and the module priority decisions. $P4$ is the Dendritic Cell Lifespan, which affects the rate at which a newly received input becomes a module, i.e. its rate of validation.

During the trials that follow, we found that the priority suppression parameter $P2$'s range of values caused very poor performance above 1.0; above which the multiplier becomes an increase rather than a reduction. We have therefore separated $P2$'s range as $P2$-*Reduced* between [0.1,..1.0] and $P2$-*Full* between [0.1,..2.0]. We will observe the behaviour of both, but are keenly interested by the intuitive *Reduced* range for analysis only.

Parameter	Default	Testing Range
$P0$	0.75	$[0.25, 0.33, 0.5, 0.66, 0.75, 0.9, 1.0]$
$P1$	2.0	$[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]$
$P2$	0.95	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.25, 1.5, 1.75, 2.0]$
$P3$	60	$[1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20, 40, 60, 80, 100]$
$P4$	50	$[1, 5, 10, 20, 50, 40, 60, 80, 100]$

Table 8.1 – Table shows the discrete testing range of values per parameter, as originally defined in 5.12, and the constant default parameter value while other parameters are varied.

8.2.2.2 Extended Metrics

In these tests seven additional global metrics are used, on top of the three metrics described in 6.2. We call these seven metrics, *Extended Metrics*. All ten metrics cover four areas: quantities of modules, time to distribute, transmissions and responses and are summarised in Table 8.2. These measures give a fuller view on architecture behaviour and performance.

Three global module quantities are measured in these tests, the first is the M1 quantity of responding detectors or state (e) in Table 6.1. We add the logged quantity of inputs read-in from the dataset, state (a), to see whether the input read-in rate is affected by the parameter configuration changes. We also add the quantity of inputs that became detectors, state (d), to discover if the detector creation rate is affected. State (d) gives the added bonus of showing the maximum possible value for M1 or state (e), by showing the maximum number of modules at one of the nodes. Similarly, state (a) gives the maximum quantity for state (d). All three metrics are network-wide measures.

Four measurements are taken with respect to the transmissions, first are the quantity of transmission send events and secondly the quantity of data sent for all transmissions. These are separated into the network wide totals and the median host. The median host results show impact on a localised scale of the exemplar node, while the totals show the network-wide impact. As used in previous experiments, the total network-wide quantity of megabytes sent is our metric M3.

Two measures of response quantities are reported, corresponding to state (h) in Table 6.1. Like the transmissions, these are the network-wide quantity and the median host's quantity of responses. These enable us to monitor if the rate of responding is affected by the parameter value changes. This can be viewed as a dependent measure of computational complexity and rate of distribution.

Reference	Description	Preference
Inputs	Number of specified inputs read-in at state (a)	MAX
Detectors	Number of specified detectors created at state (d)	MAX
Distributed-M1	Number of specified modules distributed at state (e), metric M1	MAX
Responses-Total	Number of responses network-wide at state (h)	MAX
Responses-μ	Number of responses by median host at state (h)	MAX
Time-M2	Time to distribute all M1 modules to entire network, metric M2	MIN
IO-Events-Total	Number of transmission send events network-wide	-
IO-Events-μ	Number of transmission send events by median host	-
DataSent-M3	MiB of data sent measured network-wide, metric M3	MIN
DataSent-μ	MiB of data sent measured on median host	MIN

Table 8.2 – Table of metric descriptions and references used in the architecture parameter range tests. The states refer to the state of the representation of an input, as Table 6.1. Refer to 6.2 for the metric M* descriptions. Preference is our preferred metric value, i.e. lower (MIN) or higher (MAX) values and '-' unimportant.

8.3 Results

The following are the results of varying the parameter values in isolation to answer our test objectives. In the appendix, Table F.6 and Table F.7 show the complete set of summary statistics of metrics for each parameter value change. The box plots in figures F.1, F.2, F.3, F.4 and F.5 visually show the performance of each metric, with respect to each parameter's range of test value.

8.3.1 Part 1: Parameter Effects

In this section we investigate the metric results given by testing each parameter's range of values. We answer whether the metric results are monotonically correlated to an independent parameter and which parameter directly results in the greatest impact upon performance.

To identify the correlations, we compare the ranks of median results of a metric to a set of increasing numbers with the non-parametric Spearman's *Rho* statistical test (Spearman, 1904). The impact is then measured by comparing the mean difference between each parameter value's corresponding median result of each metric.

The Spearman's two tailed test is used starting with a standard critical alpha value of $\alpha = 0.05$ and without correcting for ties. No correction leads to no additive value to *rho* in the event of equal values, which is applicable when our test purpose is for monotonicity with permitted equal values. Equal values are acceptable when a small distance between parameter values leads to a zero-to-small distance in metric space, providing the values are monotonic over the entire range. In the appendix, tables F.3, F.4, F.5, F.9 and F.10 show the *rho* statistic and *P-value* results of each ranked test. Table 8.3 shows a summary of these tables reporting the correlations with their significance, defined by $P < \alpha$.

The direct comparison of impact per parameter is given by the mean average differences between consecutive median metric results, as a result of the parameter's range value tests. In the context of the first parameter, the difference from the first median metric result and the second (and so on) gives a set of differences. The mean of those differences is taken to show the impact of that parameter on that metric. The mean difference of the metric results for each parameter can then be compared. Larger absolute mean difference values show larger average impact on the metric. The impact results

are shown in Table 8.4.

Both sets of table results show that the *Inputs* and *Detectors* metrics are unaffected by the parameter value changes, so we will not discuss them. The following is an analysis and discussion of the metric results with respect to each parameter.

Metrics	Parameters: P0	P1	P2-Reduced	P3	P4
Inputs	No effect	No effect	No effect	No effect	No effect
Detectors	No effect	No effect	No effect	No effect	No effect
Distributed-M1	Sig. <0.05 Pos.	Sig. <0.05 Pos.	Sig. < 0.05 Neg.	Sig. <0.05 Pos.	Insig.
Response-Total	Sig. <0.05 Pos.	Sig. <0.05 Pos.	Insig.	Sig. <0.05 Pos.	Insig.
Response-μ	Sig. <0.05 Pos.	Sig. <0.05 Pos.	Sig. < 0.05 Neg.	Sig. <0.05 Pos.	Insig.
Time-M2	Sig. <0.05 Neg.	Sig. <0.05 Neg.	Sig. < 0.05 Pos.	Insig.	Insig.
IO-Events-Total	Sig. <0.05 Pos.	Insig.	Insig.	Sig. <0.05 Pos.	Insig.
IO-Events-μ	Sig. <0.05 Pos.	-	No effect	Sig. <0.05 Pos.	Insig.
DataSent-M3	Sig. <0.05 Pos.	Sig. <0.05 Pos.	Sig. < 0.05 Neg.	Sig. <0.05 Pos.	Insig.
DataSent-μ	Sig. <0.05 Pos.	Sig. <0.05 Pos.	Sig. < 0.05 Neg.	Sig. <0.05 Pos.	Insig.

Table 8.3 – Summarising table of Spearman's rank (monotonic) correlations between each parameter's increasing range of values to the median results of each metric listed. The critical alpha for each two-tailed test is 0.05. Significance markers are based upon the P-values in the earlier test tables. "Pos." refers to positive (increasing) correlation and "Neg." refers to a negative (decreasing) correlation.

Metrics	Parameters: $P0$		$P1$		$P2$-Reduced		$P3$		$P4$	
	$\bar{x}$	(*Std.*)	$\bar{x}$	(*Std.*)	$\bar{x}$	(*Std.*)	$\bar{x}$	(*Std.*)	$\bar{x}$	(*Std.*)
Inputs	0	0	0	0	0	0	0	0	0	0
Detectors	0	0	0	0	0	0	0	0	0	0
Distributed-M1	**1.83**	1.7	0.429	0.623	-0.222	0.478	0.312	0.726	0.125	0.82
Response-Total	**325**	287	34.8	73.6	-9.44	167	8.75	66.3	5.31	99.8
Response-μ	**25.8**	23.4	9.21	9.02	-3.72	9.17	2.12	11.6	1.44	8.23
Time-M2	**-4.99**	6.42	-1.16	2.32	0.542	2.21	0.00375	2.42	0.0111	1.52
IO-Events-Total	47.5	48.5	0.286	2.49	0.167	1.58	7.72	25.5	-0.25	2.11
IO-Events-μ	9.83	9.51	0	0	0	0	1.62	5.62	-0.125	0.781
DataSent-M3	2.82	1.38	0.7	0.362	**-1.42**	1.37	0.525	1.28	0.025	0.363
DataSent-μ	0.567	0.281	0.136	0.0515	**-0.289**	0.238	0.1	0.265	0.00625	0.0583

Table 8.4 – Table shows the mean average ($\bar{x}$) and standard deviation (*Std.*) of (difference) impact upon each increase of parameter value. Preferred impacts are in bold. Minimum values on *Time-M2* and *DataSent-M3* metrics are preferred. Greatest values on other metrics are highlighted.

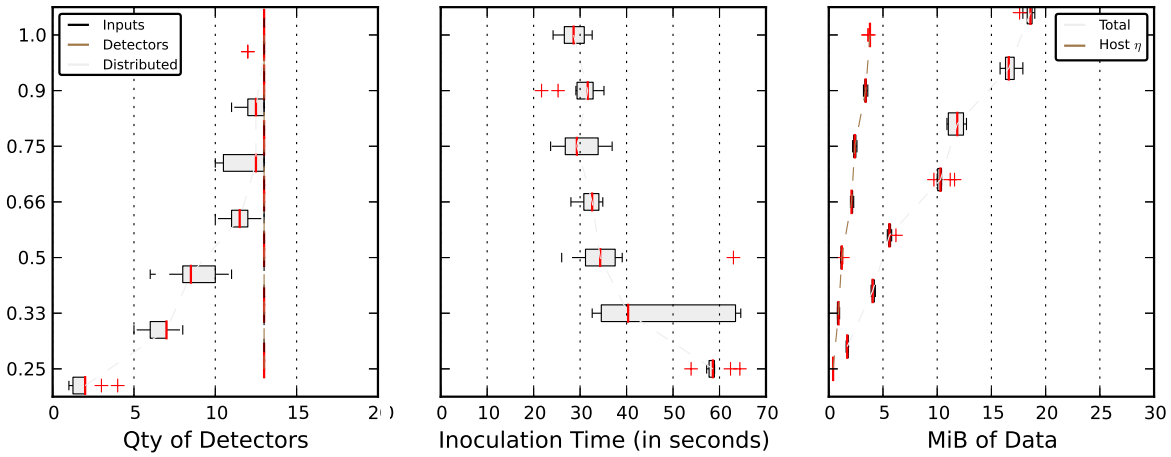


Figure 8.1 – Box plots showing metric performance while varying the under attack volume parameter (P0) value within the discrete testing range.

8.3.1.1 Under Attack Volume Percentage: $P0$

Over the discrete range of $P0$'s values, Table 8.3 shows us that *Time-M2* is negatively monotonic with respect to an increase in $P0$ configuration value. Whereas all other metrics increase with an increase of $P0$, with exception of *Inputs* and *Detectors*. The effect on the *IO-Events-* metrics will have been directly caused by a quantity change in destination hosts when the under attack threshold has been exceeded, given by Equation 5.6. Under that condition, the quantity of destinations positively correlates to the increasing parameter value.

Figure 8.1 emphasises the median metric result trajectories of *Distributed-M1*, *Time-M2* and the *DataSent-M3*, showing that the larger values give more stable and better performance in these first two metrics. *DataSent-* increases as a consequence. However, there are no apparent pivot points in the parameter's results.

The under attack volume percentage has the greatest desirable impact on *Distributed-M1*, *Time-M2* and the quantities of responses, as shown by emboldened values in Table 8.4. Thereby showing that a change in the $P0$ parameter results in the largest improvement in performance for those metrics. The increase in the *Response-Total* metric is relatively minor, as shown by an equivalent increase percentage of 0.04% of 74696.5, the parameter's maximum median value. The *Distributed-M1* increase of 1.83 shows a much larger improvement, at 14.1% of the 13 maximum distributed modules. The *Time-M2* metric's average impact is -4.99 seconds, marking a 8.5% average reduction. We also note that $P0$ carries the largest undesirable impact upon the *DataSent-M3* metric, as shown by the mean increase of 2.82 megabytes (MiB) per change. This represents a mean average increase of 15.2% in data sent of 18.6 MiB, the maximum median result for this parameter as found in Table F.6.

8.3.1.2 Initial Priority Value Multiplier: $P1$

As the parameter value of $P1$ increases, a similar set of ordinal correlations occur as $P0$. The *Time-M2* metric negatively correlates, whereas the *Distributed-M1* and *DataSent-M3* metrics positively correlate. The *IO-Events-* metric reports insignificant order correlation on the quantity of transmission events. The $P1$ multiplier has no influence on the volume selection decision, described in 5.10.1, nor the input

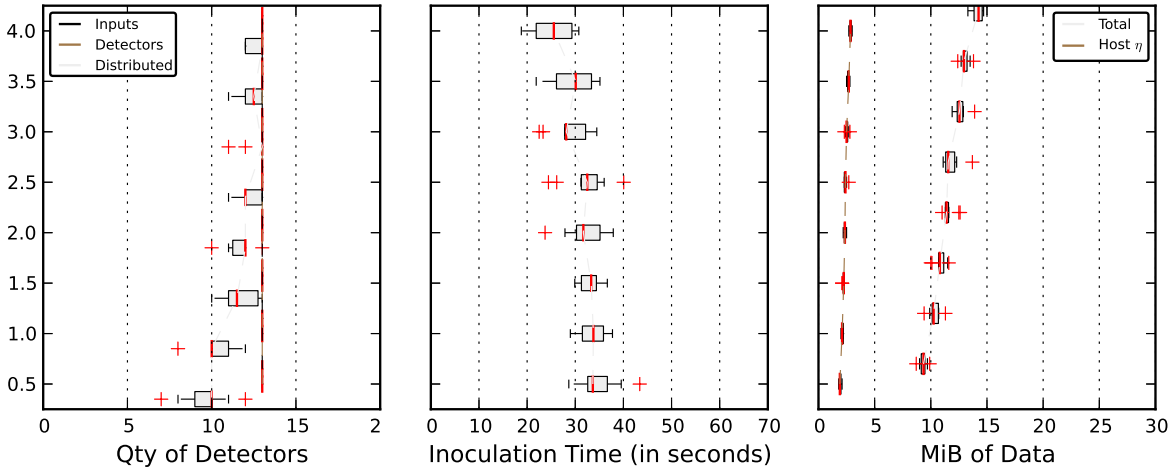


Figure 8.2 – Box plots showing metric performance while varying the initial priority multiplier parameter (P1) value within the discrete testing range.

danger value in Equation 5.3, which is likely to cause the lack of impact on the *IO-Events-* metric.

Although $P1$ shares similar directional correlations to $P0$, the impact under each metric is smaller. Each of the sample average values shown in Table 8.4 are smaller than $P0$'s impact values. Therefore we can suggest that adjusting P1 will similar effect but less relative impact on our desired performance. Figure 8.2 also visually reflects the matching ordinal correlations of *Distributed-M1*, *Time-M2* and *DataSent-M3* to $P0$. We can see that the difference in metric results between test range values are much smaller than in $P0$, while the variation of each result remains similar.

8.3.1.3 Priority Value Suppression: $P2$

Figure 8.3 shows that interpreting the impact and correlation with the *P2-Full* range of values is misleading due to a pivot point at $P2 = 1.0$. For values above the $P2 = 1.0$ pivot point, i.e. where no deprioritisation occurs, the results show very poor performance. These analysis tables show only the *P2-Reduced* range results. The figure shows that values from 0.8 and below all lead to best performance in *Distributed-M1*. Over this region of values *Time-M2* remains similar. However, there is a distinct arc of change in the amount of data sent during each of these tests.

In Table 8.3, $P2$ is the only parameter where *Time-M2* metric positively correlates to the increasing $P2$ value. The *Distributed-M1*, *Response-* and *DataSent-* metrics reduce as the parameter value increments. $P2$ is the only parameter to show the opposite correlation effect compared to $P0$ and $P1$, with respect to metrics for data sent, time taken, responses and distributed modules. This means that tuning $P2$'s value will counter $P0$ and $P1$'s effects.

In Table 8.4, $P2$ reports the best desirable impact on the *DataSent-M3* metric, unmatched by any of the other parameters. However, each increase in $P2$'s value has the most undesirable average impact on the responses, distributed modules and time taken to distribute metrics of any other single parameter change. Albeit each absolute impact is small relative to the maximum median results: at 1.71% (0.222 of maximum 13) for *Distributed-M1*, at 0.013% (9.44 of maximum 74695) for *Response-Total* and 1.69% (0.542 of maximum 32.1575) for *Time-M2*, each taken within the *P2-Reduced* range.

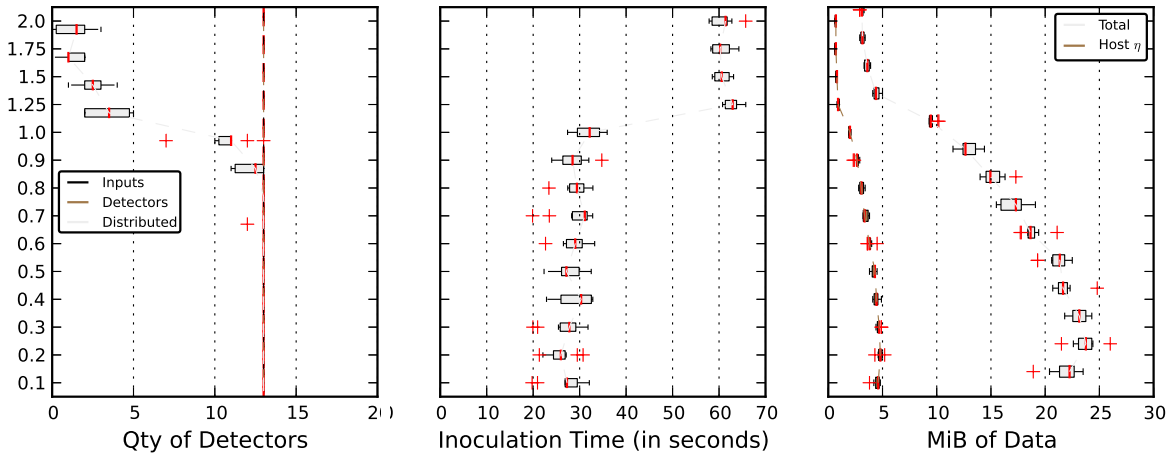


Figure 8.3 – Box plots showing metric performance while varying the priority suppression multiplier parameter (P2) value within the discrete testing range.

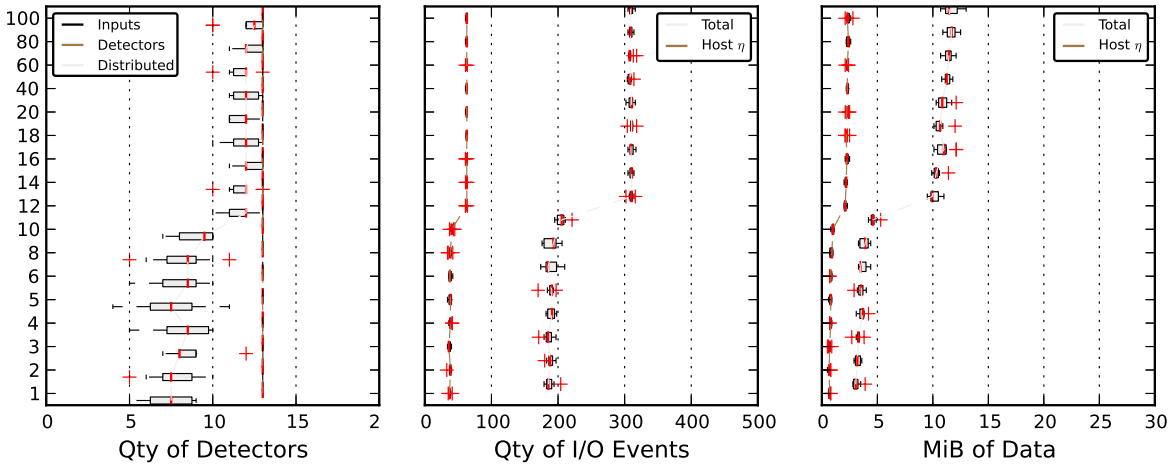


Figure 8.4 – Box plots showing metric performance while varying the static moving window size parameter (P3) value within the discrete testing range.

8.3.1.4 Static Moving Window Size: $P3$

Under our test scenario, $P3$ is the only parameter to show an insignificant correlative effect on the *Time-M2* metric while improving the *Distributed-M1* metric. As with $P0$ and $P1$, increasing $P3$'s value carries the undesirable result that increases the *DataSent-M3* metric. The impact of $P3$ in Table 8.4 shows each value change has a smaller absolute mean average difference than $P0$ and $P1$ on every metric except for the *IO-Events-** metrics.

Figure 8.4 shows a pivot point at $P3 = 10$, or with 5 seconds window duration. The effects of this pivot upon the *IO-Events-* metrics is most obvious, showing a surge in the number of transmissions made. The surge is likely to represent a pivotal value where the accumulated danger value exceeds the danger threshold, within the volume selection decision 5.10.1. Thus, this pivot point is likely to reflect the test set-up's rate of reading inputs from the model and the frequency of danger classifications in the dataset. Whether this conjecture is correct or not, the *Distributed-M1* and *DataSent-M3* metrics each show a distinct behaviour shift at this same point.

8.3.1.5 Dendritic Cell Lifespan: $P4$

Table 8.3 shows that $P4$'s value test range reports no significant monotonic correlations. The mean differences in Table 8.4 on parameter $P4$ show small values of impact, none of which measure as greatest when compared to the other parameters. This suggests that our test range for the dendritic cell (DC) lifespan parameter used in 5.11.3 has little impact and shows no ordinal effect. The appendix box plot results in Figure F.5 have no obvious behavioural trends within the $P4$ test range.

8.3.2 Discussion of Results - Part 1: Parameter Effects

This analysis has shown which parameters affect the trade-off between *DataSent-M3* and *Distributed-M1* and the *Time-M2* metric. It has shown that a change in the under attack volume percentage parameter $P0$ will have the largest single impact on measured performance of each of the metrics and that parameter $P2$ partially counters the $P0$ and $P1$ effects.

The moving window size parameter $P3$, which is a direct dependent of the under attack volume decision and priority heuristic decisions, has only a small impact on each of the metric results. The sharp surge in $P3$'s metric results noticed at 10 seconds is, we think, caused by our specific experiment design and use case on our virtual network. We expect the pivot point in other use cases to be different and dependent on the read-in rate and the frequency of danger. A fixed duration with an adaptive threshold may provide the under attack volume decision with greater robustness.

The priority suppression multiplier parameter $P2$ has an interesting range of performance between 0.1 and 0.8, leading to equal *Distributed-M1* performance and an arc effect upon the *DataSent-M3* metric as Figure 8.3 shows. The range above 1.0 leads to very poor performance and is not recommended.

$P4$ does not report any significant monotonic correlations, nor shows the greatest impact on any of the metric results. This may be due to the DC decay rate's minimal importance within the architecture considering the small validation parameter values that we have selected in 5.8.3. It may also be due to the limited testing range of values for $P4$. More likely, is that this minimal effect is caused by a lack of scaling upon the value of $P4$, or a lack of reduction scaling applied in Equation 5.15 where the $P4$ value is reduced by a fixed value. The quantity of agent interactions increases as the architecture learns. If the value of $P4$ or its reduction rate scales to match the quantity of agents, the decay rate is likely to show a greater impact and robustness.

Next, in section 8.3.3 we will report on the best performing configuration value of each range, which will help us choose a suitable value for each parameter while using the current configuration set of parameter values.

8.3.3 Part 2: Best Immunisation Rate Performance

In Part 2, we answer which parameter set-up gives the best and worst immunisation rate performance based on evaluations using the default parameter values as stated in Table 8.1; from a local viewpoint of the entire parameter value space. Having found the best configuration, we can answer whether changing that parameter's value from its default leads to a difference worth making.

Using the summary statistics shown in the appendix in tables F.6 and F.7, we can produce a set of system performance measure results for the architecture's immunisation rate. The first immunisation rate is O1 and evaluates the quantity of detectors with time taken to distribute those detectors to give a data transmission-agnostic measure of the architecture's performance. Objectives O2 and O3 incorporate the *DataSent-M3* metrics as well to evaluate two different types of networks. The immunisation rate for low-throughput networks (O2), such as SCADA networks, adds the data metric with no weighting. Primarily, O2 is the measure we are most interested in. The immunisation rate for high-throughput networks (O3), such as enterprise networks, combines the amount of data sent with a 10% weighting. Scores can range from positive (better than) to negative (worse than) the reference benchmark, where 0 is equal to the reference. The interval distance between two scores can be compared as ratios relative to a reference result. In these tests the reference benchmark has the metric results set at $[20, 3, 25]$ for *Distributed-M1*, *Time-M2* and *DataSent-M3*. In section 6.9 our three system performance measures were defined in detail, then in 6.10.3 our multi-objective evaluation functions were explained. These give us performance scores for our use cases by combining the metric results using difference intervals of ratios against a benchmark result.

Tables 8.5, 8.6 and 8.7 show best and worst immunisation rate performance under each different parameter configuration value. These entries are copied from Table F.11 where the median and standard deviation results are shown with the best statistics emboldened per parameter range. Figure F.6 visually shows the median trends with interquartile ranges of each configuration test over the 10 trials, separated by each immunisation rate objective. Figure 8.5 shows an overview of the median results per configuration per immunisation rate. Matching the analysis earlier in the chapter, we shall evaluate the *P2-Reduced* range, rather than the *P2-Full* range of values; other parameter ranges remain unchanged.

8.3.3.1 Generalised Immunisation Rate (O1)

The immunisation rate evaluation (O1) score balances the quantity of detectors with time taken to distribute those detectors, while being agnostic to the impact upon size or quantities of transmissions.

Within the O1 columns of Table F.11 we find that the greatest median objective scores are given in order by [$P1$, $P2$, $P4$, $P3$ and finally $P0$]. While considering the *P2-Reduced* range instead of *P2-Full*, the worst scores values arise from [$P0$, $P3$, $P1$, $P2$ and $P4$]. These are shown respectively with configuration values and their scores:

8.3.3.2 Immunisation Rate for Low-Throughput Networks (O2)

The immunisation rate for low-throughput networks (O2) score incorporates the amount of data sent with the immunisation rate (O1). Decisions made based on these evaluations are applicable to networks where the architecture's impact on network traffic must be low, such as in SCADA networks.

Best Scores		
Parameter	$v.$	μ Score
$P1$	4.0	-3.96
$P2$	0.2	-4
$P4$	20	-4.28
$P3$	80	-4.38
$P0$	1.0	-4.44

(a)

Worst Scores		
Parameter	$v.$	μ Score
$P0$	0.25	-9.72
$P3$	2	-5.57
$P1$	1.0	-5.38
$P2$	1.0	-5.08
$P4$	80	-4.81

(b)

Table 8.5 – Best and worst configuration values ($v.$) per parameter ranked by O1 median (μ) score.

Best Scores		
Parameter	$v.$	μ Score
$P1$	4.0	-3.74
$P2$	0.2	-3.96
$P4$	20	-4.02
$P3$	80	-4.11
$P0$	0.75	-4.3

(a)

Worst Scores		
Parameter	$v.$	μ Score
$P0$	0.25	-9.25
$P3$	2	-5.13
$P1$	1.0	-5.09
$P2$	1.0	-4.77
$P4$	80	-4.54

(b)

Table 8.6 – Best and worst configuration values ($v.$) per parameter ranked by O2 median (μ) score.

The O2 columns in Table F.11 the greatest objective scores are given by [$P1$, $P2$, $P4$, $P3$ and $P0$]. The worst score values for O2 arise from [$P0$, $P3$, $P1$, $P2$ and $P4$]. These are shown respectively with configuration values and their scores:

8.3.3.3 Immunisation Rate for High-Throughput Networks (O3)

The immunisation rate for high-throughput networks (O3) score incorporates the data sent metric with a small weighting with the immunisation rate (O1). Decisions made based on these evaluations are applicable to networks where the architecture's impact on network traffic is of lower priority than its security, such as corporate enterprise networks.

Within the O3 columns in Table F.11 the greatest objective scores are given by [$P1$, $P2$, $P4$, $P3$ and $P0$]. The worst score values for O2 arise from [$P0$, $P3$, $P1$, $P2$ and $P4$]. These are shown respectively with configuration values and their scores:

8.3.3.4 Significance of Default to Best Changes

Next is to discover if the configuration with the best scores are significantly better than their default counterparts. First we take the metric results from both configurations. Then enter the best sample set

Best Scores		
Parameter	$v.$	μ Score
$P1$	4.0	-3.94
$P2$	0.2	-4
$P4$	20	-4.26
$P3$	80	-4.35
$P0$	1.0	-4.43

(a)

Worst Scores		
Parameter	$v.$	μ Score
$P0$	0.25	-9.67
$P3$	2	-5.52
$P1$	1.0	-5.35
$P2$	1.0	-5.05
$P4$	80	-4.79

(b)

Table 8.7 – Best and worst configuration values ($v.$) per parameter ranked by O3 median (μ) score.

Parameter	U	df	***P-value***	**Significance** < 0.05
Generalised Immunisation Rate O1				
$P0$	34.0	9	0.241322	insig.
$P1$	7.0	9	0.001315	sig. < 0.05
$P2$	21.0	9	0.031209	sig. < 0.05
$P3$	12.0	9	0.004586	sig. < 0.05
$P4$	46.0	9	0.791337	insig.
For Low-Throughput Networks O2				
$P0$	2.0	9	0.00033	sig. < 0.05
$P1$	43.0	9	0.623176	insig.
$P2$	0.0	9	0.000183	sig. < 0.05
$P3$	16.0	9	0.01133	sig. < 0.05
$P4$	43.0	9	0.623176	insig.
For High-Throughput Networks O3				
$P0$	42.0	9	0.57075	insig.
$P1$	14.0	9	0.007285	sig. < 0.05
$P2$	35.0	9	0.273036	insig.
$P3$	12.0	9	0.004586	sig. < 0.05
$P4$	46.0	9	0.791337	insig.

Table 8.8 – Table showing the significance of difference between best and default parameter value configurations. Results are shown for two-tailed corrected Mann-Whitney tests (U) with an $\alpha = 0.05$ with 9 degrees of freedom (df).

of metric results into the evaluation function with the reference benchmark set as the median default result. This is followed by the default sample set with the median default result again as reference. We do this to avoid a three-point comparative analysis, between best, default and the arbitrary reference.

The two output sample sets of performance results are then passed into a Mann-Whitney U statistical test, thus making no assumption of the samples' population distributions (Field & Hole, 2003)[p246]. In Table 8.8 we have employed a corrected one-tailed test using the SciPy library. It's P-value becomes two-tailed by applying the adjustment $P * 2$. We set the critical value of alpha at the typical value of 0.05 (Field & Hole, 2003). Where $(P * 2) < \alpha$, we conclude that the populations of the two performance result samples are significantly different, thus marking an improvement to make.

The Mann-Whitney U test results of immunisation rates data in Table 8.8 show consistent significance of P-value differences between default and best parameter value configurations for parameter P3 from default 60 to best 80 (40 seconds). The reader may note that under each of the three immunisation rate evaluations (O1-3) the same best values were selected for each parameter (P0-4) respectively.

Under the generalised immunisation rate (O1) evaluation it is clear that the parameter P2 change from default 0.95 to best 0.2 leads to a significant improvement, as does the change of P1 from 2.0 to 4.0. Under the immunisation rate for low throughput networks (O2) evaluation we find that changes in P3, P2 and P0 from 0.75 to 1.0 led to significant performance increases. For the high throughput networks (O3) evaluation changes to only P3 and P1 from 2.0 to 4.0 led to significant performance improvements.

8.3.4 Discussion of Results - Part 2: Best Immunisation Rate

Based upon these results analyses we can decide that the independent changes to parameters P0, P2 and P3 will be most appropriate for our application on low throughput networks, such as SCADA networks; leading to a final configuration set of: [1.0, 2.0, 0.2, 80, 50]. It is interesting to note that

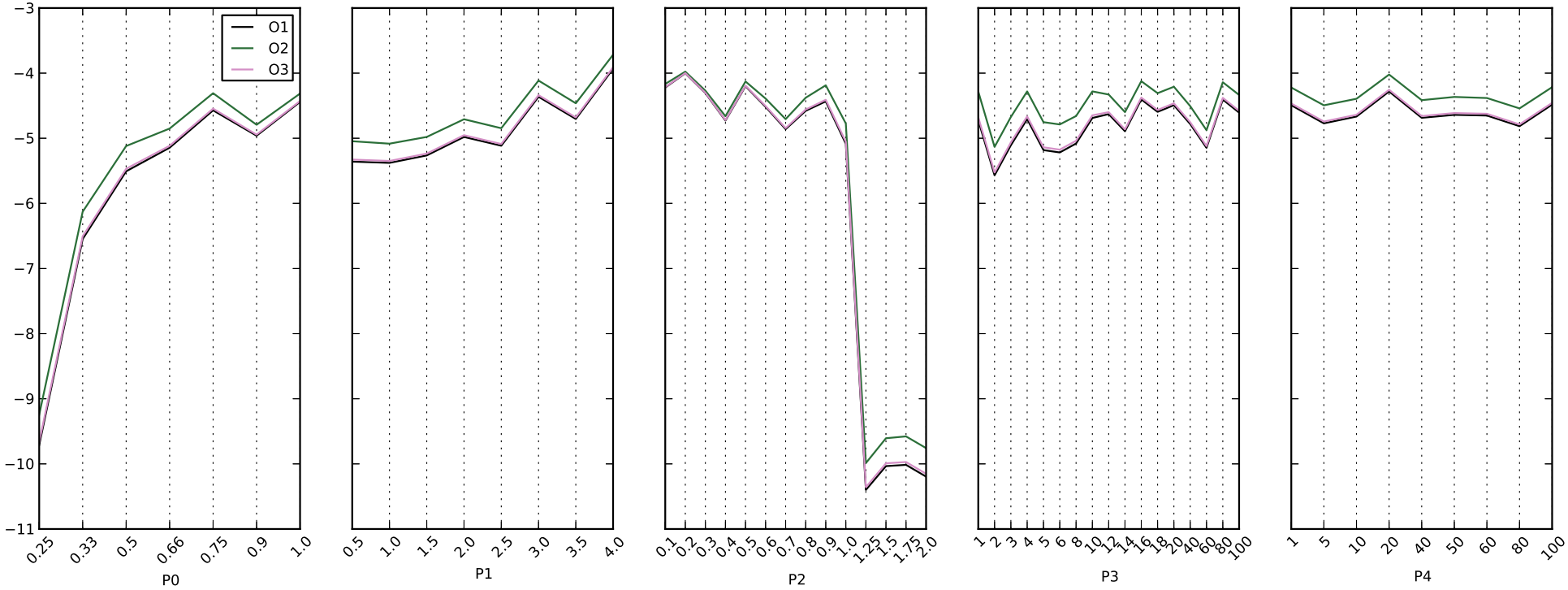


Figure 8.5 – Figure shows median results from our three system performance measures O1, O2 and O3 (scored on the *Y-axis*) from 10 trials with each parameter's set of configuration values (*X-axis*). Greater scores values are preferred. The plots from left to right show parameters: P0, P1, P2, P3 and P4, each are described within the chapter. Default parameter configuration values were $[0.75, 2.0, 0.95, 60, 50]$ respectively.

P0 had the greatest impact per interval but did not lead to the greatest overall performance. We can hypothesise that an improved configuration could be determined by improving each parameter, based on these tests. We can also hypothesise that the performance behaviours under different network conditions would be similar, this will be explored in the following items of work.

8.3.5 Conclusions

Part 1's study analysed the correlation of parameter range to metric score and discovered which parameters had the greatest impact on performance within the tested ranges. The Under Attack Volume Percentage parameter $P0$ had the greatest impact on metric performance. We found that the Priority Value Suppression parameter $P2$ reduces the time taken to transmit, that both $P0$ and $P1$ increase. The time dependent Static Moving Window Size parameter $P3$ showed a pivotal point at 10 seconds within its range of values. Above which the transmission sizes and quantities surged upward as caused by the architecture's Under Attack decision threshold. The specific value of 10 seconds we believe is distinct to our experiment set-up. Parameter $P2$ showed an interesting arced curve upon the *DataSent-M3* performance and sweet spot region between values 0.1 and 0.8. The Dendritic Cell Lifespan parameter $P4$ appears to have no significant influence of the architecture performance, which we believe to be caused by its lack of state dependent scaling, either on its threshold value or on its reduction rate of decay.

In Part 2's study we analysed the best performing parameter values and found the statistically significant best configuration set from this local point in parameter space. Based upon our performance evaluation score to measure the architecture's immunisation rate for low throughput networks we found that parameters $P0$, $P2$ and $P3$ were suitable to be changed from their defaults to maximise the performance score. For the parameters $P0, P1, P2, P3$ and $P4$ this led us to the configuration set [1.0, 2.0, 0.2, 80, 50] best suited for test using the O2 objective for low-throughput networks. The analysis shown in Table 8.8 also informs us of which parameters will most positively affect the performance of our other network type evaluations, including those for high throughput networks where data throughput impact is not a deciding factor with a high priority.

This analysis is grounded by the assumptions that the experiment set-up can be replicated to real network types of differing topology and configuration to our own virtualised network. The tested network size was small with five nodes. The loopback device provided instant and never dropped packet transmission. The high quantity of threads vying for processor time will have had an effect upon the thread execution order over the trials, thus a partial impact on execution behaviour. Most pressingly is our static default configuration set that anchored the each test. These are items of our consideration when drawing meaning from these results and lead us to toward a global parameter tuning search.

8.3.6 Next Steps

Next we will report on global searches through the parameter space for the best performing configurations under different network scenarios.

8.4 Parameter Tuning on Virtual Networks

8.4.1 Objectives

In this study we are interested to find the best configuration for the architecture to maximise its performance under our target network scenario. In this case we execute a search through a discrete parameter space and evaluate each configuration set with our immunisation rates.

A standard technique for sensitivity analysis is Latin hypercube sampling (M. D. McKay, 1979), where an equal density of samples are taken per column and row of the parameter space hypercube. The Spartan is a recent sensitivity analysis sampling software tool developed to show the relationship between a simulation and a biological system (Alden *et al.* , 2013; Alden *et al.* , 2016). As CARDINAL-Vanilla is an interaction directed algorithm there is a real possibility of extreme fluctuation in metric score results as the parameter values change. To reveal whether this is indeed the case an exhaustive approach is needed; however to progressively improve throughout the space, a constrained exhaustive step-wise hill climber is used.

8.4.2 Experiment Design

These tests followed the same experiment design as earlier in this chapter while running on the environment configuration defined in 7.2.4. The quantity of network nodes remains at five with 10 repeated trials and the time desynchronisation set at 0.45 seconds, identically to 7.2.4 and as analysed in 7.6.3. The differences from the virtual network benchmark tests are the dataset and the wrapping of the search around the architecture's experiment set-up.

The execution script glued together the architecture's usual execution cycle into the architecture evaluation module, the search algorithm's selection module and the new configuration generation modules. These integrations led to the automation of the search, which is described in more detail below. The architecture user input model used the CSIC Dataset D, see 6.7.1.

8.4.3 Search Method

The search method employs a hill climber with a step-wise constrained-exhaustive search as illustrated in Figure 8.6. In theory the hill climber cannot be guaranteed to find a global optimum. However, a singularly elitist stepwise exhaustive search gives different benefits. It enables visualisation of the parameter behaviour from multiple points in parameter space, thereby gaining information about parameter sensitivity and likelihoods of best performance regions per parameter range. The stepwise re-evaluation of the same ranges will also permit enquiry into the effects of uncontrolled (stochastic) operational noise, although this remains as open work. Of course exhaustive searches are time expensive, due to this the stopping criteria is set at 10 generations. After re-evaluating the same parameter ranges ten times from different points in search space we expect to have tested a relatively good configuration, based upon our results containing factorised noise.

The first generation takes the default parameter configuration set shown in Table 8.9 with a sets of configuration files for each of the parameters, $P0$ to $P3$. Each configuration file has a single change from the generation's initial default set. Each is evaluated under the experiment set-up. After all of these configurations have been evaluated, the best configuration contains a single changed parameter

value from the initial default set and is selected for the next generation. This is the step-wise hill climb. The next generation's configurations are generated. New configuration files for each parameter, $P0$ to $P3$, are created as shown in Table 8.9, with the newly updated parameter defaults. The execution and evaluation of these new generation configurations then begins.

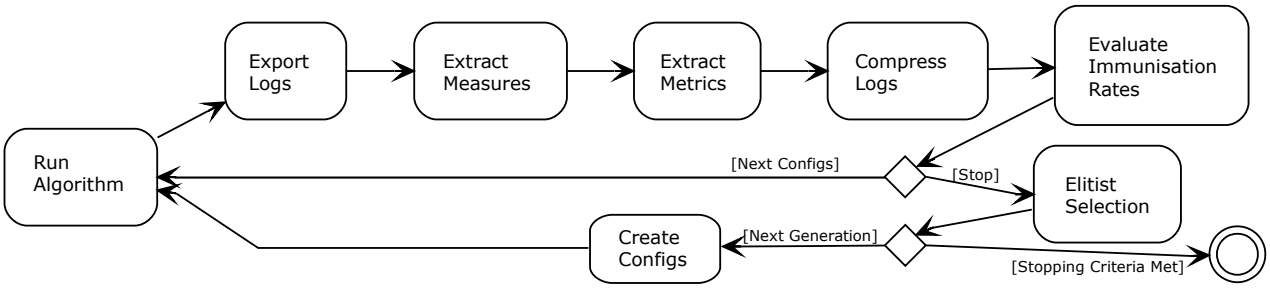


Figure 8.6 – Diagram showing the optimisation pipeline employed to tune the parameter configuration.

The elitist selection process selected a best single configuration. The immunisation rate for high-throughput networks (O3) was used to evaluate the system performance of each configuration. This O3 score directed the search. At the time of testing this evaluation objective was preferred in initial preparation for enterprise network testing. Below we present each of the immunisation rate scores including those for low throughput networks (O2).

8.4.4 Parameter Tuning Ranges

These tests use reduced ranges of values per parameter as depicted in Table 8.9. The first of the changes from the previously defined ranges are upon the parameter $P4$'s range which has been removed as its effect upon immunisation rate performance was found to be insignificant in 8.3.3.4. Secondly the *P2-Reduced* range replaces the *P2-Full* range as it gives superior performance and is more semantically intuitive, as noted in 8.2.2.1. Thirdly the $P3$ range is reduced to limit the discrete search space. The $P3$ range of 1 to 5 was selected at the time due to empirical belief that $P3$ had only a small effect upon the immunisation rate scores. The total search space has 2,800 states and each generation will evaluate thirty of those states. Therefore approximately 10% of the space will have been evaluated after ten generations with ten independent steps through the search space from the default set.

Parameter	Default	Testing Range
$P0$	0.75	$[0.25, 0.33, 0.5, 0.66, 0.75, 0.9, 1.0]$
$P1$	2.0	$[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]$
$P2$	0.95	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$
$P3$	4	$[1, 2, 3, 4, 5]$
$P4$	50	$[50]$

Table 8.9 – Table shows the discrete testing range of values per parameter, adapted as a result of the earlier tests upon the original ranges. First generation default parameter value are shown.

8.5 Results

Part 1's results analysis shows the best configurations outputted during the search method as each generation completed. The criteria for best configuration is the evaluation of the immunisation rate for

high throughput networks (O3). As we have seen before, O3 combines three metric scores *Distributed-M1*, *Time-M2* and *DataSent-M3*. Each of the 10 trial runs of a given configuration give the three measurements. In Part 1's analysis, the median measurement of each metric from the 10 trials was selected to represent the configuration. The Ratios with Inverses equations described in F.1.1 evaluate that set of three metric scores. The O3 score of each configuration is put forward into a pool. The best configuration in the pool determines the next generation's default parameter set. Table 8.10 shows the score of the best measured configuration after each search step.

In Part 2's results analysis the best discovered configurations are reported, based on the Ratio of Distances evaluation equations (in section 6.10.3). Part 1's Ratios with Inverses equations inaccurately combine metrics with a MIN-type preferred semantic condition. To remove the effect of this inaccuracy on the encountered search candidates each is re-evaluated using the accurate Ratio of Distances evaluation measure. Ratios with Inverses equations are critiqued in section F.1.1. Figure 8.7 illustrates the double evaluation of each candidate configuration set (search state) as analysed within Part 1 and Part 2.

In Part 2, the network-wide metrics have been re-extracted to give the complete set of *Time-M2* recorded results. The standard deviations and interquartile ranges for *Time-M2* have been calculated.

Tables 8.11 and 8.12 show the best ranked configuration sets from all of the search states. The ranking of each configuration set is given by its median immunisation rate score for low throughput networks (O2). In this part's analysis we consider no restrictions on the source generation of the best rankings.

The best ranking result sets for the other two immunisation rate scores are given in the appendix. See tables F.12 and F.13 for the generalised immunisation rate (O1) and see tables F.14 and F.15 for the immunisation rate for high throughput networks (O3).

8.5.1 Part 1: Search Results by Generation

The best runtime evaluated score is shown at generation 9 led by the runtime evaluation of the O3 score for high throughput networks. By generation 9, the search configuration had maximised the generalised immunisation rate (O1); the rate that does not account for data transmitted. At generation 9 the elite configuration had maximised the value of $P0$ and minimised $P3$ and $P2$ values, with $P1$ at a low-to-medium value position within the range.

The best immunisation rate scores for low and high throughput networks (O2 and O3) of all the generations were found in generation 4. Its elite configuration minimised $P2$ and $P3$, while keeping a medium-high value for $P0$ and a medium value for $P1$. Generation 4 showed the best median and standard deviation on *Distributed-M1*, 4th-best *Time-M2* median score and 4th-best *DataSent-M3* median, standard deviation and interquartile range scores. Generation 3 and 4 show the same configuration with slightly differing performance scores, from this we draw two conclusions. First this configuration takes higher preference due to taking the equally highest mode of the elite configurations, along with generations 1 and 2. Second is the variation between any two sets of trial runs of the algorithm is affected by noise giving metric scores; in this case 1.48 standard deviations for *Distributed-M1* and 0.36 standard deviations for *DataSent-M3*.

The elite configuration with the best *Time-M2* score came at generation 7, however its value is relatively similar to generations 4, 8 and 9. We did not record enough data during runtime search to

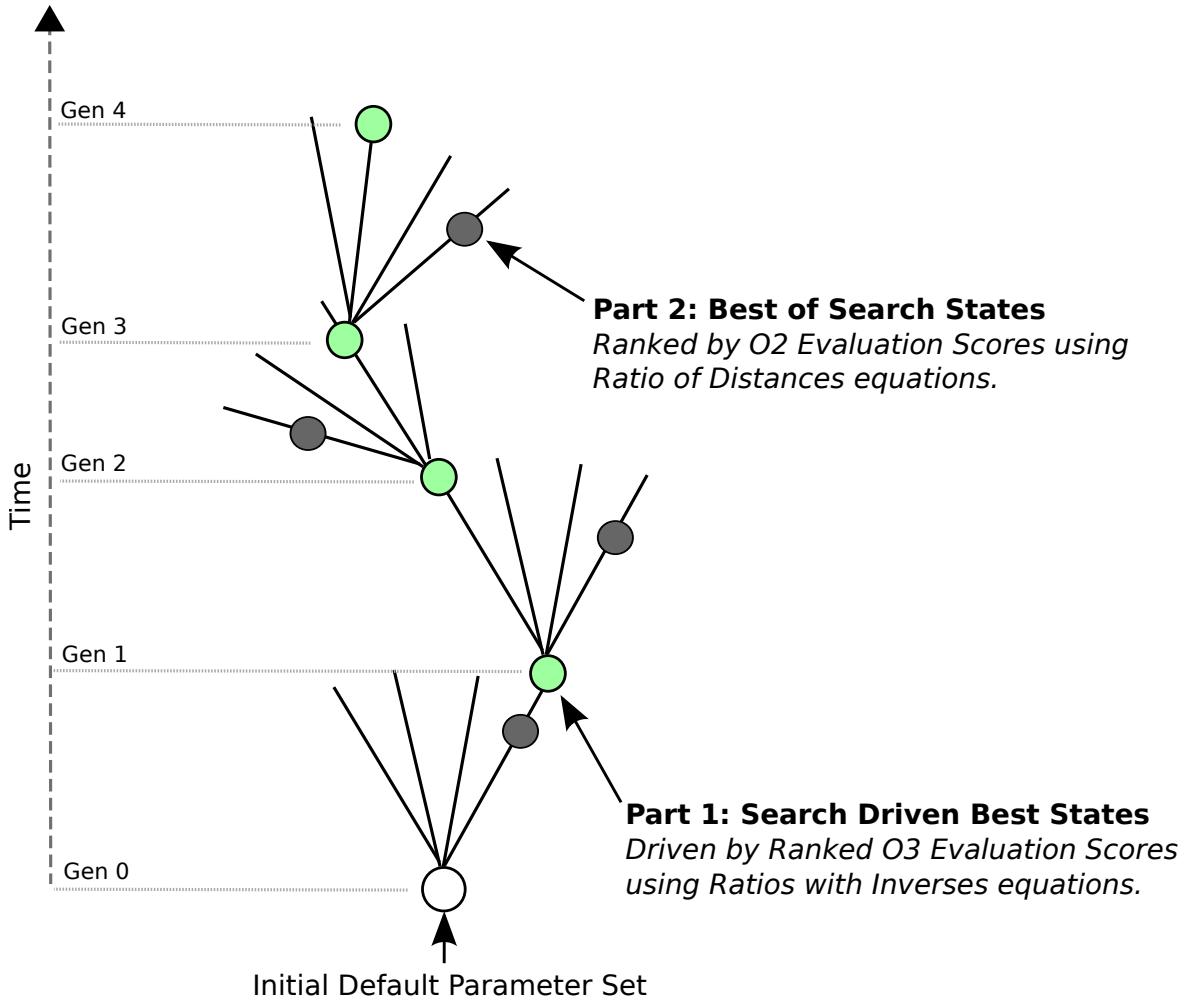


Figure 8.7 – Illustration of the search states analysed within the text. Highlights the differing immunisation rates and evaluation equations used during the analysis. Green circles represent the configuration set selected during the step-wise search. Grey circles represent the configuration set selected post-search, using the Ratio of Distances evaluation equations. Four lines from each circle represent values along the four parameter dimensions.

evaluate the standard deviation or interquartile range of *Time-M2* thus are unable to show significance. However, the metric's median scores show the elite configurations are closely grouped. The *Distributed-M1* median scores show most of the elites perform similarly, with generation 4 showing lowest standard deviation. The generation with the best *DataSent-M3* median score came at generation 3, which matches the configuration at generation 4.

8.5.2 Discussion - Part 1: Search Results by Generation

In this study we found a configuration set that gave the best immunisation rate performance on both low and high throughput networks (O2 and O3). Generation 4's unique configuration set was selected as elite the most number of times during the search, together with a very similar set at generation 2 and gave the best O2 and O3 scores of the entire search. The search led to a configuration that should have maximised performance for high throughput networks; however, it actually found configurations that finally increased the amount of data sent leading to a final generation with the best generalised immunisation rate (O1) score.

Over the 10 generations the search selected configuration that minimised both $P3$ and $P2$. We also notice there is some relationship between increasing $P1$ and lower $P0$, as seen at generations 5 and 6. The following search step recovered to the more expected maximisation of $P0$, which in turn lowered the value of $P1$. Over each generation, we also notice that the proximity of each of the median metric scores are close. The standard deviations between two sets of trials (generations 3 and 4) on the same configuration are within 1.48 and 0.36 standard deviations for *Distributed-M1* and *DataSent-M3* respectively. Thus, while there most certainly is a noticeable difference between a bad and good configuration, between the best configurations there are only relatively small differences.

The next analysis will report on metrics collected by a repeated extraction of measurements from the runtime log files with the complete set of *Time-M2* metric results per configuration and evaluated using the Ratio of Distances equations, stated in 6.10.3.

8.5.3 Part 2: Best Search States

The top ranked configuration set lowers the $P0$ value and results in a large jump in the generalised immunisation rate (O1) and the high throughput rate (O3) scores. Rank number 1's has the best *Time-M2* score which is likely to be the reason for its performance hike. It has the 2nd-best *Distributed-M1* score and the 9th-best *DataSent-M3* score, although seven of the other ranks' median *DataSent-M3* scores are within 1.26 standard deviations, see ranks 3,4,5,6,7,9,10. Rank 1's $P0$ value of 0.66 is found in 30% of the top 10 ranked configuration sets and its $P1$ value of 0.1 is found in 90% of the top ranks. $P0$ with a value of 0.66 is not found in any of Part 1's elite configuration sets, as a result of it having a different immunisation rate to optimise.

Part 1's best configuration set (0.75,2,0.1,1) for immunisation rate score for low and high throughput networks (O2 and O3) appears twice and is ranked at 2nd and 6th place in this new ranking. The rank six set was selected from generation 4 which matches the best O2 set found in Part 1. The rank two set was found in generation 5, which did not appear as an elite set in Part 1.

The best *Distributed-M1* score is given by ranks 6 and 10, overall this ranking shows fewer configuration sets maximising the number of distributed detectors score. The best *Time-M2* score came from rank number 1. The best *DataSent-M3* came from rank 8's configuration set. Ranks 8 and 10 share

Generation	Configuration Set P0	P1	P2	P3	Metric Scores: Distributed-M1 μ	Std	IQR	Time-M2 μ	Std	IQR	DataSent-M3 μ	Std	IQR	Immunisation Rate Scores O1	O2	O3	Parameter Change Param	Config	Value
init.	*0.75*	*2*	*0.95*	*80*	*19.0*	*1.676*	*0.75*	*17.112*	-	-	*61.1*	*9.448*	*14.4*	*-1.29297*	*-1.93759*	*-1.35743*			
0	0.75	2	0.95	2	19	3.247	0	16.707	-	-	52.8	15.387	18.775	-1.27571	-1.81501	-1.32964	P3	C1	2
1	0.75	2	0.1	2	**20**	0.516	1	18.156	-	-	53.65	11.371	10.85	-1.29871	-1.84953	-1.35379	P2	C0	0.1
2	0.75	2	0.1	2	**20**	0.699	0.75	16.394	-	-	51.2	5.826	8	-1.22507	-1.74217	-1.27678	P1	C3	2
3	0.75	2	0.1	1	19	0.675	0	16.883	-	-	**42.65**	9.153	6.325	-1.28327	-1.66858	-1.32180	P3	C0	1
4	0.75	2	0.1	1	**20**	0.483	0.75	15.787	-	-	46	4.869	4.425	-1.19785	**-1.63770**	**-1.24184**	P3	C0	1
5	0.75	3.5	0.1	1	**20**	1.265	0.75	16.44	-	-	45.45	8.036	11.275	-1.22709	-1.65826	-1.27021	P1	C6	3.5
6	0.5	3.5	0.1	1	**20**	0.699	0.75	17.452	-	-	44.35	10.511	17.425	-1.27018	-1.68368	-1.31153	P0	C3	0.5
7	1	3.5	0.1	1	19.5	1.826	1	**15.346**	-	-	61.5	15.966	10.4	-1.19568	-1.84501	-1.26061	P0	C6	1
8	1	1	0.1	1	**20**	3.719	1	15.439	-	-	60.55	14.890	17.85	-1.18177	-1.81987	-1.24558	P1	C1	1
9	1	1	0.2	1	**20**	0.707	1	15.397	-	-	67.7	17.081	23.65	**-1.17981**	-1.89842	-1.25167	P2	C1	0.2

Table 8.10 – Table showing the best configuration sets ranked at runtime by their *O3* immunisation rate score over the ten search generations running under the virtual network testing environment. Greatest immunisation rate scores (O1,O2,O3) are preferred. Preferred result values are shown in bold. The median *Time-M2* values were runtime decision factors used to calculate O1,O2 and O3, these are reported. The standard deviations (*Std*) and interquartile ranges (*IQR*) for the *Time-M2* metric are missing as they were not extracted at runtime.

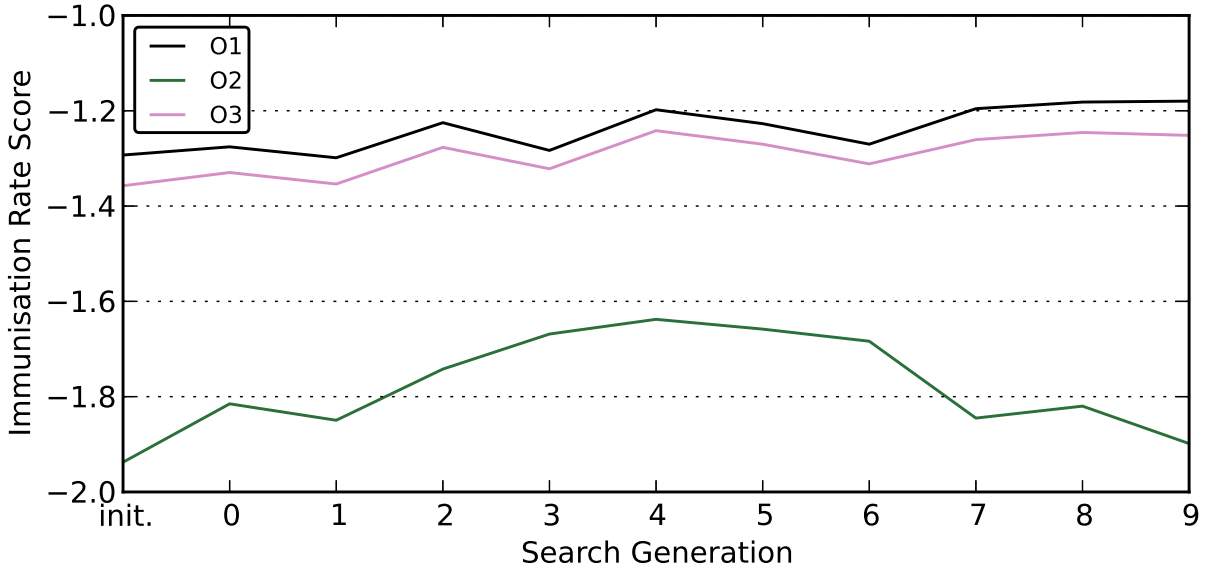


Figure 8.8 – The plot shows the median immunisation rates (O1,O2,O3) from the table above over the 10 search generations.

the same configuration set with $P0 = 0.66$ and $P1 = 3.5$ and take a 66% share of best metric scores. However, they rank lower as they share the two lowest *Time-M2* scores of the ranking.

There is large standard deviation on the rank 3 configuration set. This is most noticeable in the *Time-M2* metric score and then subsequently in the standard deviations of the immunisation rate scores. With a deeper understanding of the *Time-M2* metric, the implemented code and the test condition we can conclude this as follows. In the event where one of the network nodes fails to receive any detectors, i.e. due to failing to open its receiving communication socket, and that node locally read-in zero desirable detectors, then the *Time-M2* score is set at the duration of the entire test. This larger value is instead of the usual median time for detector distribution. Other variations on this case can lead to a similar result. Therefore, we consider this an unrepresentative result of the configuration and instead make our judgements based upon its median scores and their interquartile ranges.

8.5.4 Discussion - Part 2: Best Search States

In this analysis study we found a new top ranked configuration set within the discrete testing range defined in Table 8.9, based on the immunisation rate for low throughput networks (O2). The top ranking set lowered its value of $P0$ and raised its $P3$ value based upon Part's top ranking set for O2, which led to a better *Time-M2* performance rate. We also found supporting evidence for the best elite configuration set from Part 1. Part 2's rank 2 and rank 6 configuration sets match Part 1's best set for low and high throughput networks (O2 and O3) from generation 4. Both Part 1 and Part 2's top configuration sets for O2 are relatively similar – sharing $P1$ and $P2$ parameter values and within 1 step for both parameter $P0$ and $P3$. Based upon the results of these analyses, either configuration sets are good initial candidates for selection in similar future applications.

This analysis has shown that lowering the parameter $P0$ value of % of hosts and detectors to distribute to a point at or around 0.66-0.75 leads to the best performance for the immunisation rate scores for low throughput networks (O2). While minimising the priority suppression parameter $P2$ gives best performance. In these tests we have found that lowering the time dependent parameter $P3$ gives best performance, but we note that it has some relationship to $P0$ and is likely to have a dependency on the available hardware resource and throughput capacity; in this case the time dependent bottleneck is the virtual network and the machine running the test.

We have found that many of the generations have given a top 10 ranked set. This implicates that the timing of a generation, – i.e. a day, where a single generation ran for approximately 24 hours, – may have had only a small effect on the immunisation rate O2 score; however this is an unmeasured statement.

8.5.5 Conclusions

This parameter tuning evaluation has found two candidate parameter configuration sets for the CARDINAL-Vanilla architecture that maximise the performance for SCADA like networks, those networks that require a lower impact upon network throughput. The underlying test configuration used our virtual network environment with five nodes, as such the direct application of these candidate configuration set should be used with an understanding of the ordinal effects of each parameter, described within, and an understanding of the effects of decentralised distributed applications running on the target network hardware and topologies. The two top ranked parameter configuration sets for our application led us

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	0.66	2	0.1	2	19.500	0.500	1.000	**14.889**	2.246	2.835	46.650	3.939	7.150	2
2	0.75	2.0	0.1	1	19.000	0.640	0.000	16.078	2.729	0.734	38.750	6.573	8.775	5
3	0.75	4.0	0.1	1	19.000	5.869	2.250	16.819	64.784	3.944	41.650	6.109	5.550	4
4	0.75	3.5	0.1	1	19.500	1.166	1.000	16.653	6.946	4.079	48.800	6.536	4.800	6
5	0.75	2	0.95	1	19.000	1.414	1.000	16.109	4.779	2.835	42.550	13.512	15.100	0
6	0.75	2.0	0.1	1	**20.000**	0.458	0.750	15.712	1.368	2.486	46.000	4.619	4.425	4
7	0.75	3.0	0.1	1	19.500	0.663	1.000	16.478	1.484	2.693	45.950	5.040	8.375	6
8	0.66	3.5	0.1	1	19.000	2.202	0.000	17.498	4.936	3.758	**37.100**	3.951	4.075	7
9	0.75	2.5	0.1	1	19.500	0.663	1.000	16.715	1.387	0.947	44.600	7.602	7.950	6
10	0.66	3.5	0.1	1	**20.000**	0.490	1.000	16.993	2.099	2.489	44.450	7.915	8.425	7

Table 8.11 – Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the virtual network environment set-up.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	0.66	2	0.1	2	**-1.99392**	0.37643	0.46621	**-2.48217**	0.40510	0.49479	**-2.03912**	0.37866	0.47236
2	0.75	2.0	0.1	1	-2.21717	0.46889	0.12229	-2.51475	0.53059	0.32529	-2.23610	0.47378	0.11767
3	0.75	4.0	0.1	1	-2.32008	10.93923	0.71133	-2.52467	10.89624	0.70050	-2.33988	10.93487	0.71836
4	0.75	3.5	0.1	1	-2.28800	1.18286	0.68617	-2.57450	1.18113	0.71717	-2.31653	1.18204	0.67610
5	0.75	2	0.95	1	-2.19733	0.82436	0.44750	-2.60208	0.86355	0.81237	-2.23218	0.82438	0.48117
6	0.75	2.0	0.1	1	-2.11858	0.22656	0.42683	-2.64167	0.23447	0.34158	-2.17048	0.22566	0.41126
7	0.75	3.0	0.1	1	-2.24633	0.24584	0.45504	-2.66658	0.30767	0.56258	-2.28203	0.25089	0.45179
8	0.66	3.5	0.1	1	-2.44125	0.87496	0.64508	-2.66742	0.83518	0.56692	-2.46185	0.87074	0.64313
9	0.75	2.5	0.1	1	-2.32325	0.23102	0.13908	-2.67758	0.17613	0.24271	-2.35422	0.22148	0.14260
10	0.66	3.5	0.1	1	-2.33858	0.35360	0.41787	-2.69492	0.40379	0.57225	-2.38688	0.35578	0.42862

Table 8.12 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the virtual network environment set-up.

to a preferred set at or near (0.66,2.0,0.1,2) and (0.75,2.0,0.1,1) for parameters $P0$ through to $P3$. A real life application of the architecture may find good relative performance from a new configuration set that is based on the top two sets; i.e. found by interpolating values 0.66 and 0.75 for parameter $P0$ and values 1 and 2 for parameter $P3$, where other parameter values can be taken directly. The ranked parameter configuration set evaluations for the alternative high throughput networks are shown in the appendix.

8.5.6 Next Steps

This test's environment set-up used a number of nodes that we expect to be smaller than our target application and its virtual network environment is known to behave differently than in conventional networks. Therefore, our next item of work is to analyse the parameter configuration performance under an enterprise network set-up, together with a larger number of nodes.

8.6 Parameter Tuning on Enterprise Networks

The architecture will have to handle different conditions of the network either by adapting to or selecting a preferable architecture parameter configuration. The objective will be to maximise the architecture's immunisation properties without hindering the network or computational performance of the underlying machines. Before installing the architecture on the corporate workstations of an ICS or SCADA network, we must find a preferable parameter configuration.

This study will search for the best configuration for the architecture under our enterprise network experiment set-up. Matching the earlier parameter tuning on our virtual network, we will execute a search through a discrete parameter space and appraise each configuration with our immunisation rate evaluations.

8.6.1 Experiment Design

These tests followed the same experiment design as used in the enterprise benchmark tests in 7.4.3.1, with the workstations' environments set as in D.2. Twenty wired network workstations were used in each test. In this experiment the dataset read-in by the distributed user model was the *v0.5.2* CSIC dataset version, described in 6.8.

The execution script wrapped the architecture's experiment framework into the parameter tuning framework, which integrates the search generation evaluation module, the configuration file generator and the scripts to manage the logs, configurations files, metric results files and immunisation rate evaluation files, as Figure 8.9 shows. The script for this parameter search is described in detail in D.4.

8.6.2 Parameter Tuning Ranges

The search uses the parameter value ranges specified in Table 8.13. The virtual network search experiment in 8.4 helped us to further refine and reduce the $P0$ and $P2$ parameter ranges giving the best performing values shown in the table. The ranges give 1800 possible states in this parameter search space.

Parameter	Testing Range
$P0$	$[0.5, 0.66, 0.75, 0.9, 1.0]$
$P1$	$[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]$
$P2$	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
$P3$	$[1, 2, 3, 4, 5]$
$P4$	$[50]$

Table 8.13 – Table shows the discrete testing range of values per parameter, adapted as a result of the earlier tests upon the original ranges.

8.6.3 Search Method

The search method in these tests used a multiple start hill climbing algorithm, similar to that described by (Mahdavi *et al.* , 2003). The change in optimisation method was due to the amount of time taken. The search procedure matches the earlier method and its pipeline is restated for clarity in Figure 8.9.

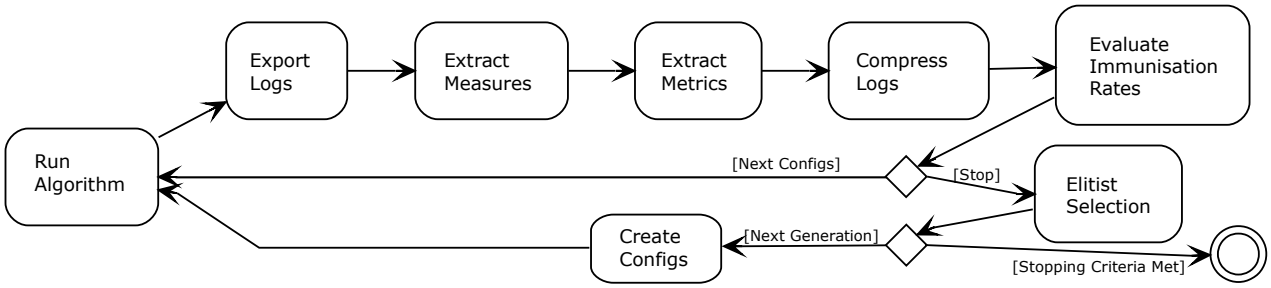


Figure 8.9 – Diagram showing the optimisation pipeline employed in the parameter configuration search under the enterprise network test set-up.

The classic hill climber search, as used in the earlier virtual network search, is prone to getting stuck in local optima. A multi-start hill climbing algorithm climbs from N randomised starting points, pursuing only a percentage of the best paths. This improves its likelihood of reaching a higher fitness than the classic climber, however neither can be guaranteed to reach the global optimum. Mahdavi et al have suggested that this search approach gives good information for subsequent searches (Mahdavi *et al.* , 2003) and as such provides an approximation of the global optima.

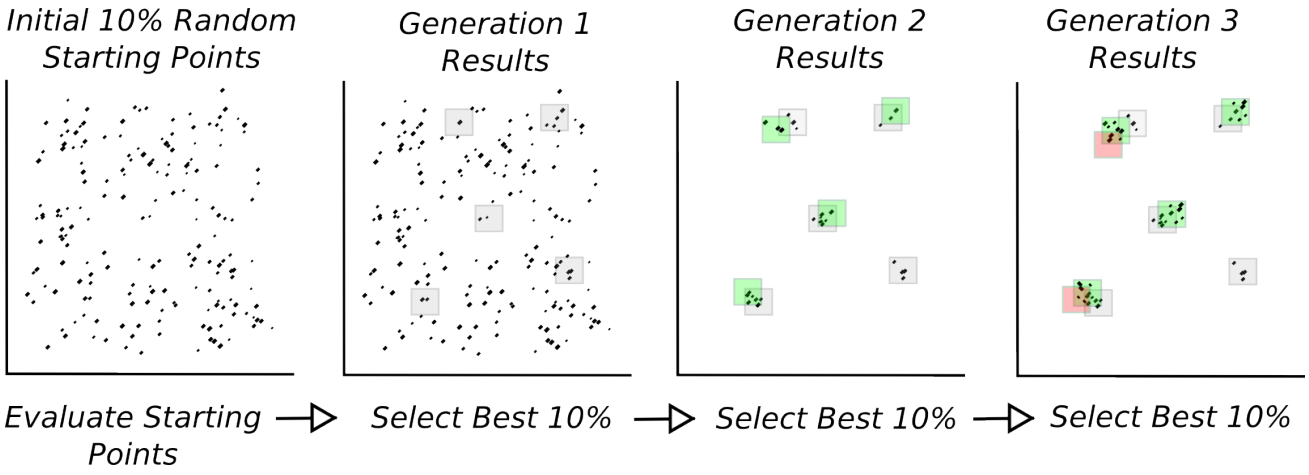


Figure 8.10 – Illustration of the search steps taken by the multi-start hill climbing parameter tuning approach.

The first generation begins with N starting configurations, randomly selected from the parameter state space. The architecture is run with each configuration over 10 trials and then evaluated. 10% of the top evaluated configurations are selected as starting candidates for the next generation. Each *elite* configuration's parameters are incremented and decremented to give eight further candidate configurations. Duplicates within this or from an earlier generation are not retested. This decision carries the assumption that performance over time will not be negatively affected by noise. The parameter increments are index-based increments within the bounds of each range of values, defined in Table 8.13.

Suitable stopping conditions are when the slope of sorted performance scores levels out, i.e. each configuration's fitness is statistically insignificantly different from its predecessor's fitness when the result set is sorted by performance, or when there are no more configurations to test. Time was a factor in our case, which meant that after 6 days of trials consisting of 3 generations and a total of 317 configurations tested the search was terminated.

$v.$	Description
1800	Quantity of configuration states.
10%	Starting points (180 configurations) randomly selected.
10%	Top n% of results selected from each generation.
O3	Selection fitness evaluation function.
20	Architecture network node quantity.

Table 8.14 – Table of the multi-start hill climbing algorithm's parameter values.

The configuration metric results of each generation are evaluated using the immunisation rate for high throughput networks (O3) using the Ratios with Inverses equations mechanism. This evaluation function selected the *elite* configurations per generation and directed the search in the same manner as the virtual network search in 8.5.1. The flaw with the Ratios with Inverses equations is it inaccurately combines metrics with a MIN-type preferred semantic condition, critiqued in F.1.1. This results section instead reports performance of each immunisation rate objective calculated using the reliable Ratio of Distances evaluation equation, as stated in 6.10.3. Under both evaluations, a fixed metric result set of $[M1 = 20, M2 = 3, M3 = 25]$ provided the preferred reference benchmark against which the ratios were calculated.

8.7 Results

The results presented consist of the best 10 configuration sets ranked by their immunisation rate score for low throughput networks (O2) over the 3 search generations. The ranking is calculated using the reliable Ratio of Distance evaluation equations stated in 6.10.3. Tables F.17, F.19 and 8.19 show the metric results from generations 1 to 3, while tables F.18, F.20 and 8.20 show their calculated immunisation rate results.

Figure 8.11 summarises the median immunisation rate score performance for each of the 317 configurations over the 3 generations over six days. It clearly shows differing behaviour between generations 1 and generations 2 and 3. So the challenge is to find meaning from the stochastic search.

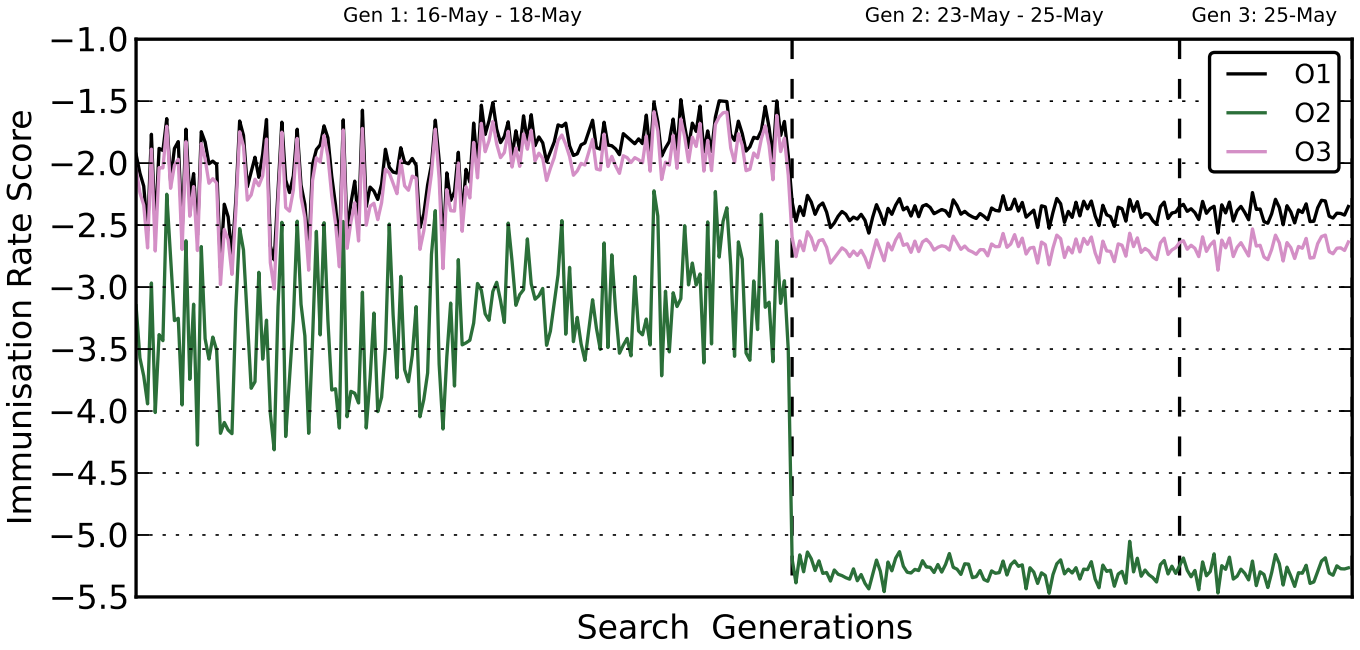


Figure 8.11 – The plot shows the median immunisation rates (O1,O2,O3) over the 3 search generations and highlights the experiment noise.

8.7.1 Reasoning through the Noise

There is variability in the min-max range of performance scores between generation 1's configurations and the subsequent generations. This is explained by the random selection of the generation 1's configuration sets and the stepwise selection used in generations 2 and 3. The reduced min-max range of variability at the mid-point in generation 1 is caused by a shift from 5 trials (A) to 10 trials (B) per configuration test leading to a more settled median in the second half of generation 1. This B result set and the subsequent B_2 and B_3 sets each use 10 trials. There is a stepped-down performance change at the start of generation 2. Shown below is evidence that indicates this is due to uncontrolled network activity. This conclusion is drawn as no other factors were changed except the configuration sets and the time/date of the tests on the network, however the quantity of packets sent has increased dramatically.

The network is managed by other staff and the switch is auto-managed, therefore quality of service (QoS) policies may have throttled the architecture transmissions or TCP:port transmissions. Dropped packets or suspended streams would cause transmission retries, slowing the *Time-M2* score and increasing the *DataSent-M3* scores. Generation 1's top results consistently show a *Time-M2* score 5 seconds less than and a *DataSent-M3* score 100 MiB less than the top results of generations 2 and 3, Table 8.15 summarises this. The value of 2.75 in the *DataSent-M3, Diff* column at generations 2 and 3 may suggest that on average packets under went 2.75 retries before successfully arriving, taking on average 34% more time to transmit. We conjecture this explains the stepped-down performance change.

Alternatively if other workstation activity was under-way, time to process the inputs would slow leading to a larger *Time-M2* and which could cause thread priority changes delaying the time at which

	Mean Metric Scores					
Generation	*Dist-M1*	*Diff.*	*Time-M2*	*Diff.*	*DataSent-M3*	*Diff.*
1	13.7	-	11.9065	-	61.215	-
2	14	1.022	16.0007	1.344	168.565	2.754
3	14	1.022	15.9732	1.342	168.695	2.756

Table 8.15 – Table shows mean metric scores per generation from the top 10 configuration results ranked by immunisation rate for low throughput networks (O2). The *Diff.* columns show the multiplier difference from generation 1, at 3 decimal places.

	***IO-Events* Median Host**					
	Gen 1		Gen 2		Gen 3	
Measure	Sent	Recv	Sent	Recv	Sent	Recv
Mean	177.714	176.088	318.982	318.860	318.982	318.814
Median	175	175	319	319	319	319
Std.	15.286	12.996	0.132	1.538	0.133	1.449
Max	236	219	319	323	319	324
Min	147	151	318	315	318	315

Table 8.16 – Table of descriptive statistics shows a clear difference between generations, based upon the number of *IO-Events*, corroborating the belief that network throttling caused the metric differences between the generations. The *IO-Events* metric quantifies the number of transmission events by the median host of the network, per trial; stats are shown over the entire range of trials per generation.

the detectors were selected. This could lead to a larger quantity of available detectors, thus larger packet sizes affecting *DataSent-M3*, but this would not explain the *IO-Events* metric showing a large discrepancy between transmission events found between generations 1 and generations 2 and 3, as shown by Table 8.16.

Thus we conclude network throttling to be the likely cause of the stepped-down performance change after generation 1. Therefore we can draw comparisons from within each generation, but not between all generations. Based upon the *IO-Events* metric results shown in Table 8.16, one could believe that the throttling, or noise, is equal upon both generations 2 and 3. Equally controlled experimental conditions mean that comparisons between Gen 2 and Gen 3 are valid, but comparisons between Gen 1 and the other generations are not valid. The search selection remains valid. Therefore the best selected configurations from all generations are valid to consider best in there respective generation; however, comparisons between the best of the three generations is explicitly not valid. In an ideal scenario we would control for any network throttling. In future experiments we would use the results of control trials to deduct from test trials in order to identify fluctuations.

8.7.2 Best Configuration Set Results

The top 10 configuration sets shown in Table F.17, F.19 and 8.19 appear, at first glance, to be non-uniform. The top 10 configurations contain parameter values from top, bottom and mid ranges for parameter $P0$, $P1$ and $P3$, whereas $P2$ broadly stays high throughout the generations. After some analysis we do find compelling probability characteristics on the parameter value regions.

The emboldened values in Table 8.17 show the percentage of each best performing parameter falling above or below its mid-point in each generation. In generation 1 we see that $P0$ and $P1$ were low in their ranges. $P2$ was high and $P3$ was low. In generation 2, $P3$ was equally distributed in its range, $P2$ remained high and $P0$ and $P1$ remained low while moving marginally toward a higher value. In

generation 3, $P0$ became higher, $P1$ stayed low and $P2$ remained high and $P3$ became higher.

We found at the end of generation 3 that $P0$ and $P1$ had similar traits to the virtual network parameter tuning search in Table 8.10, as Table 8.18 summarises. In the virtual network test $P0$ increased to its maximum of 1 in the range 0.5-1; in this test $P0$ had a value of 0.75-1 in 70% of the top 10 cases, its top ranked configuration selected the highest value of 1. In the best of the virtual tests, $P1$'s value was low to medium within the range. In the enterprise test $P1$'s value ranked low to medium in 60% of the best configurations, with a midrange value of 3 in the top configuration.

However, Table 8.18 shows us that $P2$ and $P3$ clearly performed better at the opposite ends of their parameter ranges when we compare the enterprise and virtual network test conditions. It is possible that the differing network sizes (5 and 20) cause this case, but we believe this is due to the unique time conditions of each test. $P3$ depicts the time duration of the danger definition within the architecture. $P2$ defines how quickly detector priority is diminished, larger values cause quicker deprioritisation. There is a relationship between faster suppression over a longer duration of $P3$ and very slow suppression over a short duration of $P3$. Both cases have shown good performance under the different test conditions. Ranks 3 and 9 in the top 10 B_3 result set counter this relationship by having opposing low and high values for those parameters, but do perform well. However, we are yet to find medium ranged values perform as well as the extremity cases. Further generations of this search would be unable to find

In Table 8.18 parameters $P1$, $P2$ and $P3$ show some relations. There is a correspondence between low to medium initial priorities with low priority suppression over short durations, and between medium to high initial priority with higher suppression over longer durations. We observe examples of both cases offering good immunisation rate performance for low throughput networks (O2). A critical viewpoint taken may suggest the tuning is attempting to remove the impact of the priority heuristic by equalising the components of the its uprate/down-rate regulation, this is a subject open to future enquiry.

Generation 1: Top 10 (A and B)					
P0	0.5≥	**80%**	< 0.75 ≥	20%	<1.0
P1	0.5≥	**70%**	< 2.0 ≥	30%	<3.5
P2	0.7≥	10%	< 0.8 ≥	**90%**	<0.9
P3	1.0≥	**70%**	< 2.5 ≥	30%	<4.0

Generation 2: Top 10 (B_2)					
P0	0.5≥	**60%**	< 0.7 ≥	40%	<0.9
P1	0.5≥	**60%**	< 2.25 ≥	40%	<4.0
P2	0.2≥	30%	< 0.55 ≥	**70%**	<0.9
P3	1.0≥	50%	< 3.0 ≥	50%	<5.0

Generation 3: Top 10 (B_3)					
P0	0.5≥	30%	< 0.75 ≥	**70%**	<1.0
P1	2.0≥	**60%**	< 3.0 ≥	40%	<4.0
P2	0.2≥	10%	< 0.55 ≥	**90%**	<0.9
P3	2.0≥	10%	< 3.5 ≥	**90%**	<5.0

Table 8.17 – The tables map each of the elite configurations' parameter values into either the top or bottom regions of the parameter's range. The percentages show where those configuration values fell from each of the elite configurations per generation.

Network	Size	Generation	Use Case	Parameter Region $P0$	$P1$	$P2$	$P3$
Virtual	5	At Gen 9	Directed best O3	1 (H)	1 (L)	0.2 (L)	1 (L)
Virtual	5	At Gen 4	Directed best O2	0.75 (H)	2 (M)	0.1 (L)	1 (L)
Virtual	5	After Gen 9	Best O2	0.66 (M)	2 (M)	0.1 (L)	2 (ML)
Enterprise	20	After Gen 3	Best of O2	70% (H)	60% (ML)	90% **(H)**	90% **(H)**
Enterprise	20	At Gen 3	Best O2 in B_3	1 (H)	3 (M)	0.7 **(H)**	4 **(H)**
Enterprise	20	At Gen 2	Best O2 in B_2	0.9 (H)	3.5 (M)	0.7 **(H)**	5 **(H)**

Table 8.18 – Table shows a comparison of best parameter values selected during each of the parameter tuning searches. Emboldened are the opposing findings for parameters $P2$ and $P3$ in the enterprise to the virtual parameter tuning tests. L,ML,M and H refer to low to high values within each parameter's test range.

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	1.0	3.0	0.7	4	14.000	0.000	0.000	16.188	0.807	0.816	164.800	5.719	2.250	B_3
2	0.9	3.0	0.8	4	14.000	0.600	0.000	15.848	1.299	0.599	167.050	6.038	6.950	B_3
3	0.75	2.0	0.2	5	14.000	0.000	0.000	15.983	0.725	0.449	167.050	2.945	1.175	B_3
4	0.5	2.5	0.9	4	14.000	0.000	0.000	15.931	0.755	1.461	168.000	5.609	6.450	B_3
5	0.9	2.5	0.7	4	14.000	0.400	0.000	16.093	0.699	0.882	165.550	5.295	4.875	B_3
6	1.0	4.0	0.7	4	14.000	0.000	0.000	15.524	0.909	1.128	171.550	4.175	6.125	B_3
7	0.66	2.5	0.8	4	14.000	0.000	0.000	15.897	1.291	0.828	169.550	1.965	2.525	B_3
8	0.9	3.0	0.6	4	14.000	0.000	0.000	15.950	0.936	1.829	169.950	5.108	9.125	B_3
9	1.0	2.5	0.9	2	14.000	0.300	0.000	16.248	1.684	1.508	173.150	8.404	8.300	B_3
10	0.5	2.5	0.7	4	14.000	0.300	0.000	16.070	0.768	1.371	170.300	6.673	7.975	B_3

Table 8.19 – Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_3.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	1.0	3.0	0.7	4	-2.34800	0.13448	0.13596	-5.15200	0.17103	0.13613	-2.62840	0.13425	0.12835
2	0.9	3.0	0.8	4	-2.29142	0.22988	0.09975	-5.15783	0.34258	0.25854	-2.58062	0.24080	0.11435
3	0.75	2.0	0.2	5	-2.31375	0.12079	0.07483	-5.17867	0.12462	0.17700	-2.60378	0.11988	0.07825
4	0.5	2.5	0.9	4	-2.30517	0.12585	0.24346	-5.18408	0.14575	0.12571	-2.58037	0.12347	0.22956
5	0.9	2.5	0.7	4	-2.33208	0.12168	0.16571	-5.18550	0.16887	0.23308	-2.62578	0.12316	0.16281
6	1.0	4.0	0.7	4	-2.23733	0.15156	0.18792	-5.21033	0.18212	0.26129	-2.53043	0.15285	0.20137
7	0.66	2.5	0.8	4	-2.29950	0.21524	0.13796	-5.21050	0.23073	0.20587	-2.59230	0.21652	0.14006
8	0.9	3.0	0.6	4	-2.30842	0.15606	0.30479	-5.21700	0.18655	0.28104	-2.59442	0.15640	0.31412
9	1.0	2.5	0.9	2	-2.35792	0.28184	0.27004	-5.23150	0.23056	0.34583	-2.64282	0.27252	0.28464
10	0.5	2.5	0.7	4	-2.32825	0.12558	0.21983	-5.24417	0.12751	0.11146	-2.60695	0.11923	0.19518

Table 8.20 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_3.

8.7.3 Discussion

The experiment infrastructure gave a plethora of noise hiding the comparisons over the generations while transforming the second and third generation results of the multi-start hill climbing search. By observing indicators of QoS throttling on the transmission values of our non-metric *IO-Events* test measures we drew understanding from the underlying results. Table 8.18 shows the best of the information extracted to help readers choose an appropriate configuration.

The best configurations selected after six days of trials and three search generations show some similarity and some conflict to those found in our earlier virtual network parameter tuning search. Parameters $P0$ and $P1$ have matching traits to the earlier tests. $P0$ was maximised to transmit more detectors to more hosts. The tuning pipeline assigned $P1$ a midrange to minimised value for a detector's initial heuristic priority value. More generations may further support these similarity, however due to operational needs of the network at this test scale we were unable to tune further.

Parameters $P2$ and $P3$ differed from the virtual tests by mostly raising or maximising their values within their ranges by the end of the third generation, shown by example in Figure 8.12. Earlier in the chapter, in Figure 8.5 a small dip in parameter $P3$'s performance between values 1 and 5 can be found, as is tested during this tuning. The cause of this search path difference between the virtual and enterprise tests remains unclear. We believe this is the result of an another region of good performance for the combined effect of the two parameters, however it may have been selected as a result of the differing network sizes or execution times of each experiment set-up.

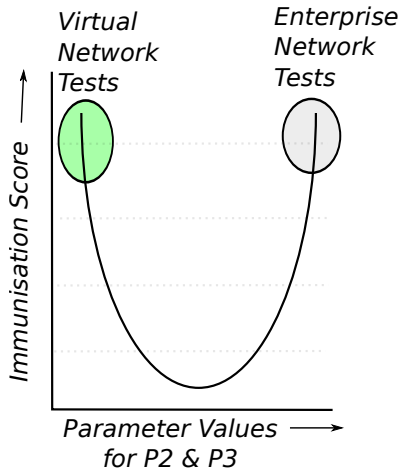


Figure 8.12 – An illustration of the saturated regions of highest immunisation rate scores found within the parameter ranges for $P2$: [0.1-0.9] and $P3$: [1-5]. Circled are the regions preferred by the virtual and enterprise network conditions.

While, an advantage of our search method was its rapidity to converge on an optima in only a few generations, it remained prone to finding locally-optimal solutions, similarly to the standard hill climb except with a lower probability. Climbing further generations would be unlikely to resolve the $P2$ - $P3$ condition. This is because performance of parameters $P2$ and $P3$ seem to be mappable to a parabolic curve, i.e. medium values gave poorer performance, extremities gave better performance – and the selected elites of generation three each have high range values. The stepwise index-increments used to modify each generation would be unable to traverse the range(s) if the performance is indeed parabolic. An alternative stochastic or sub-optima resolving search would be required.

The best performing configurations appear to give good immunisation rate results, and can be chosen as appropriate for the target network. Alternatively these configurations can stand as a base for future tunings. Without the effects of throttling we can expect the best performing generation three configurations to perform similarly to or faster than the random configurations tested in generation one. It remains open work to discover if a configuration in generation one may be optimal. Evidence from the virtual network range testing and tuning analysis suggests the generation three configurations show greater similarity to the best of those earlier tests, and are therefore our configuration sets that take our preference.

8.7.4 Conclusion

In this study we have tuned the architecture to find an good quality parameter configuration set to use with the architecture while running on low throughput networks, such as SCADA or ICS networks. Despite the operational network noise, after the tuning's third generation we have found regions within each parameter range that optimise our immunisation rate objective, as Table 8.17 depicts. Table 8.19 has presented the ranked top 10 configuration sets of the tuning after its third generation; together with Table 8.18 these configurations are the preferred configurations for this network type as selected by the tuning search under our experimental conditions. Finally, we have found similarity and conflict in the best performing configuration sets when comparing the virtual and enterprise network tuning experiments, as shown in Table 8.18. This table also summarises the best performing configurations from both tests.

8.8 Chapter Conclusions

Overall this chapter has presented a broad and detailed analysis of the architecture's key parameter performances. The first parameter range testing experiment has shown the importance and impacts of each parameter based upon a static point in parameter space. The second parameter tuning study exhaustively explored the ten dynamic points in parameter space while running under a virtual network set-up with five nodes, and gave a small number of best performing configurations for the architecture. The third study ran a parameter tuning search on the architecture while running under our enterprise network set-up with twenty workstations and results in a few elite configurations that will be appropriate for selection in real applications of this architecture.

These works have found similar immunisation rate behaviours in the Under Attack Volume Percentage ($P0$) and Initial Priority Value Multiplier ($P1$) parameter under the differing real and virtual network experiment set-ups. In parameters Priority Value Suppression ($P2$) and Static Moving Window Size ($P3$) we have observed conflicting preference by each of the tuning methods, which we believe is either caused by these parameters sharing parabolic performance curves leading to two regions of good performance or by the distinct timing differences shown between the two network types. Parameters $P0$, $P2$ and $P3$ have been shown to give the greatest impact on our immunisation rate score for low throughput networks, such as SCADA and ICS networks.

A small set of parameter configurations that show good performance under the two test network conditions have been presented in Table 8.18. A summary of the good performance regions of each parameter range are shown in Table 8.17 as found under our enterprise network of 20 machines.

8.8.1 Peripheral Discoveries

Although this chapter's work has achieved our aim of finding appropriate parameter configurations for the architecture, its further testing has highlighted other interesting findings.

The network throttling occurring during the enterprise network tuning tests has presented us a problem. While common among peer-to-peer distributed systems, it has identified for us the task of bottleneck avoidance as a feature necessary for future large scale decentralised self-healing architectures. In engineered algorithms for distributed system state transfer further state packets must be sent to request and inform other nodes of which update data is required. This adds to the quantity of transmissions. This Artificial Immune System (AIS) approach entirely avoids those additional packets and still performs well; it does so by using its priority heuristic to evaluate the importance of local data for transmission. We note that monitoring of data bitrate or transmission retries will allow nodes to adapt their performance to further fit the network throttling bottleneck and further throughput capacity thresholds.

The first two experiments have broadly covered the architecture parameter ranges and have reported good robustness over many configurations. Clearly we can find poorer performing configurations, as we have shown in 8.3.3. However there are many better performing configurations, many of which give approximately equal performance scores. Figure 8.5 is perhaps our best example of this, showing that our selected ranges of the parameters $P1$, $P3$ and $P4$ perform well. Within the confined ranges both $P0$ and $P2$ report good performance and poorer performance depending on the value regions.

The results of the decay parameter $P4$ during the first study showed insignificant effect which was surprising. This caused us to analyse further the likely behaviours of the dendritic cell (DC) agent's lifespan and its decay within the Multi-Agent System (MAS) component of the architecture. From that analysis we reached an expectation that the decay rate and lifespan length lacked in scalability over time and over the number of objects within the multi-agent system. Thorough testing and updating of this component remains as open work from the viewpoint of the AIS architecture.

8.8.2 Future Work

As an adaptive reconfiguration test for the architecture, the tuning test set-ups take a long time to complete, and do not show huge rewards providing an initially good configuration is selected. Additionally they are not nearly the most interesting component to adapt. Locating and appeasing or circumventing bottlenecks in the system, such as the QoS throttling that we experienced in the enterprise network tuning search are in themselves challenges for adaptive and self-healing software systems.

We can mitigate network throttling and even recognition by targeted attacks by monitoring and varying the intervals, quantities and protocols of the architecture's transmissions. By performing periodic input analysis and feature selection we can select the most informative features to aid better classification decision making. This can lead to improved anomaly recognition as the sensor input definition of normal changes over time. We can also vary the graph connectivity map used by each architecture node. If we monitor the connected architecture devices for latency, data bitrates (upload/download) and the switch routing paths then the software can adapt to changes in the network conditions and topologies over time.

These kinds of pervasive applications of intelligence as mechanisms for adaptivity can be trialled and must comprise software engineered frameworks of the future; be their application to anti-malware / anti-intrusion tasks or for more widely applicable applications requiring robust underlying distributed system platforms, such as Apache's Hadoop[1] or TypeSafe's AKKA framework[2].

[1]The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. https://hadoop.apache.org/

[2]Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the Java virtual machine. http://akka.io/

Chapter 9

Towards a Decentralised Self-Healing Security System for Industrial Networks

This chapter presents a future work solution to our problem case, that of a decentralised self-healing security system for industrial networks. The description contains our design principles, an in-depth view of the self-healing system component applied to industrial networks and takes a broader view of self-management components needed in self-healing security systems of the future. The applied recovery generation is specifically targeted at automation networks with Siemens S7 controllers, but the theoretical approach can be applied to any programmable logic controller (PLC) manufacturer. This architecture is named CARDINAL-E, as a conceptual extension to the CARDINAL-Vanilla self-healing architecture stated in chapter 5 and originally modelled by (Kim *et al.* , 2005).

9.1 Design Principles

The architecture's design is based upon principles and characteristics drawn from immune system cell interaction and organisational behaviours, as described in chapter 3, and partially take from the past works in the survey of distributed self-healing Artificial Immune Systems (AISs) shown in 4.2. Any implementation of this security software system should adhere to state-of-art best practices in security design principles and methodologies; reference should be made to those guidelines. Our newly acquired principles and ideas lead us to two components and categories of system behavioural goals:

Self-Healing Component Tenets:

- **Detection:** all devices will use trained models to recognise deviations in normal activity.
- **Recovery:** all devices will execute recovery modules or scripts that revert behaviour to a normal or improved operational state, guided by historical data and performance objectives. Devices performing the intensive role (i.e. behaviours of lymphocytes and in lymph nodes) will generate and validate modules.

- **Collaborative Model Training:** intensive role devices will build and distribute classifier models using collected historical device activity data and performance objectives.

Self-Management Component Tenets:

- **Redundancy:** devices that perform the intensive role will store recovery tools, historical data and resources to support other devices.
- **Social Sensing:** devices inspect each other, via issued scripts, to discover the device's current activity.
- **Self-Organising:** device roles adjust depending on the available resources.
- **Optimising Footprint:** devices adapt their processing tasks and communications, including scheduling and execution delays, to maintain a minimal resource impact and to avoid saturating system resources.
- **Regional Clustering:** devices will recognise scaling bottlenecks and will regionally cluster nodes into sub-architectures. This enables a balance of impact on communication and resources against full system redundancy and processing objectives.
- **Collaborative Decisions:** devices can issue commands upon the other devices, however a concentration threshold must be breached, i.e. a request from multiple origins, before the recipient device will act.
- **Trusted Communications:** devices will communicate using trusted session mechanisms.
- **Openness:** architecture devices will 'admit' new devices into the network by installing the client. Other devices will inspect a new device's secure state and perform repair actions.
- **Avoidance Strategies:** architecture devices will adapt their behavioural profiles to avoid malicious software targeting the architecture.

9.2 Self-Healing Component

The distributed self-healing architecture will run on each device connected to the network. The architecture will generate and serve classifier models and recovery tools to the devices. These data-driven tools will require sensor and indicator information from across the network. Not all devices will have the computational power to generate the tools and therefore a role separation is necessary. The role separation and tool storage will be redundant in a decentralised manner to avoid the single point of failure problem. Every device will have a capacity to receive and use the up-to-date tools and also will transmit its local sensor and indicator information.

A reward-penalty fitness function using the performance indicator information will support user-driven reinforcement for classifier model learning. This indicator information provides the system's application-specific definition of damage and malicious behaviour. Recovery tools are focused on industrial networks, specifically enabling an automated restore and track-back feature on the PLCs. The self-management component takes the recovery and defence mechanisms a step further and follows the self-healing component feature descriptions.

9.2.1 Role Separation

The system will monitor and learn a dynamic model of normal activities on each industrial network device including transmissions received at the PLCs and other sensor data on the network. Passing input data through a trained machine learning model can be fast, however the behavioural model learning of large amounts of information is time intensive. In a dynamic system, this intensive task is additionally more frequent.

We can make *responsible* processing use of the industrial network devices, such as HMIs, historian and acquisition servers for example. However, as these devices are often of varying age, operator usage and available resource capacity any attempt to perform intensive tasks on these devices can render them inoperable, defeating our purpose. A device's availability can be recognised by monitoring and learning its activity profile based on time-series patterns. Therefore the system needs role switching to vary the architecture's activity, depending on its devices' states.

Our view of *responsible* use is risk reduction and architectural robustness. Firstly to avoid impacting the devices' operational duties, but more subtly to avoid a single-point-of-failure of important roles of the self-healing security system. From the collaborative immunisation viewpoint, it is essential to enable each device a capacity to perform these important roles, such as to create a recovery solution to aid itself or other devices.

The separation in resource capacity will designate the *lightweight* and *intensive* performance roles of the framework. This mechanism of dividing the roles is dependent on the network devices' availability, we refer to this as self-management, performed by a self-management component within each device on the distributed system. The intensive role operates in addition to the lightweight role, however only when available capacity and timing are appropriate.

The lightweight role devices perform detection and execute recovery solutions in a lightweight manner, i.e. without carrying out a model learning or validating process. To facility the lightweight functions, these devices periodically send monitored inputs or traffic to the intensive role devices. The intensive role consolidates the network-wide contextual data for analysis and uses it to learn detection models and recovery solutions, as we will describe further below. The intensive role devices will issue these security modules to the lightweight devices in order to respond to matching input data.

9.2.2 Additional Hardware

First we add an additional functional module with dedicated processor and network interface to each PLC, such as a replacement to the Siemens S7 CP343-1* network module. We refer to this as the PLC bus module (PLCBM). Secondly we add a dedicated server and optional single purpose connection to a trusted computing cluster with dedicated processing capacity. The computing cluster is for offline model learning of the collected data and supports the self-healing functions. The other devices, e.g. human machine interface (HMI) workstations and historian servers, etc., connected to the industrial network form essential host-based viewpoints and detection components of the architecture.

The PLCBM device supports information verification by providing sensor and controller state data with greater trust than elsewhere on the network. As you will recall, the Stuxnet malware caused its infected PLCs to send industrial network packets with false sensor values to the monitoring and acquisition systems, thus avoiding detection (Falliere *et al.* , 2011). The PLCBM will act as a trusted

device on the PLC rack. Therefore, in theory at least, it can gain direct access to the sensor values stored in memory at the PLC or at the PLC's driver that receives the data from the sensor/actuator communications bus. The PLCBM can pass these values into its local classifier, i.e. the detection model trained on network-wide collected data. From our perspective, if a PLCBM reported sensor value differs from the historian reported sensor value, there is cause for investigation. Further, if other security indicators report deviations from normal and worsening performance evaluation, there is cause for action.

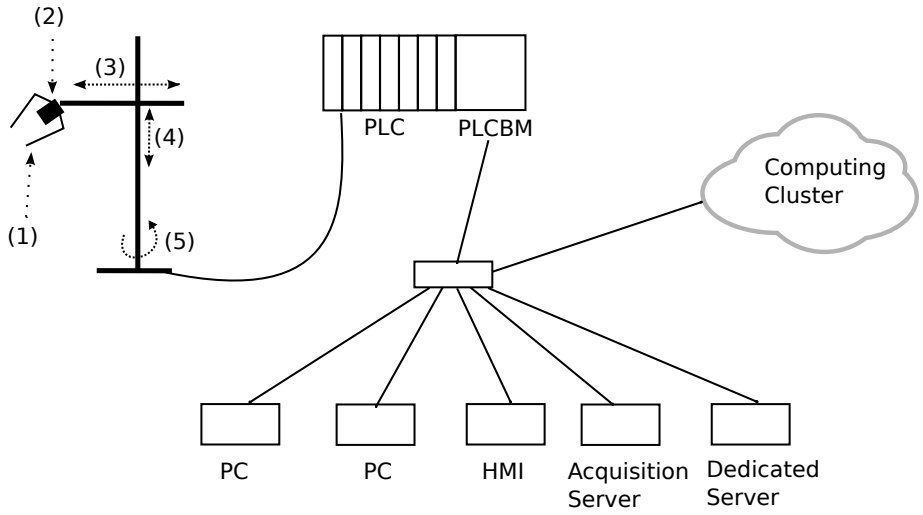


Figure 9.1 – A toy-example topology with added hardware integrated and automation sensors numbered.

9.2.3 Performance Indicators

The operational performance of the automation system, the security system and its recovery actions can be evaluated over a range of performance indicators from several areas of the network. The example indicator categories are:

- (SAI) Sensor value anomaly indicators
- (APTI) Advanced persistent threat (APT) indicators
- (PLCI) Programmable logic controller (PLC) state anomaly indicators
- (SPI) SCADA network packet indicators
- (BSI) Business success indicators, i.e. quantity and quality of product
- (VCI) Video capture of device indicators, i.e. device action, output and anomalies.

To correlate these indicators to our example topology in Figure 9.1, the SAI indicator are values from the automation sensors (1-5) collected directly by the PLCBM and collected by the acquisition server via the standard SCADA network packet request-return process. The PLCI are to monitor the PLC state as seen by the PLCBM, acquisition server, and other developer workstations; these indicators include functional block checksums and timestamps. The APTI monitor activity indicative of connections to/from the network and irregular activities on the devices. These items are well studied and range from application-layer firewalls, to process call tree analysis, to audit logs of incorrect password alerts and beyond. The SPI are to monitor the communications to the automation system. SCADA network

protocol-specific intrusion detection systems and firewalls on the industrial network will provide SPI indicator measurements. The BSI measure performance from the business or organisational perspective. Measurements over output quantity, output quality and maintenance costs, for example, can indicate undesirable effects of the current automation system state. The VCI monitors anomalies in automation system's visual behaviours. This measurement can indicate anomalies in the physical activity of the automation equipment.

Our selection of indicator categories is led by the belief that a single indicator cannot be guaranteed to pinpoint the root cause of a novel attack. However, the combination of relevant and informative indicators that measure belief in our system objectives and the good operational state of our system can localise a novel attack's acting location or targeted system. We believe the specific objectives be driven to avoid damage to the automation system and its production output, while the specific measurements must be selected for the specific use case.

9.2.4 Belief and Objective Weightings

An initial set of weights are assigned per indicator category to each neighbour device. If the device can provide the indicator measurement, it is assigned a weight value of 1, otherwise it is assigned a weight value of 0.

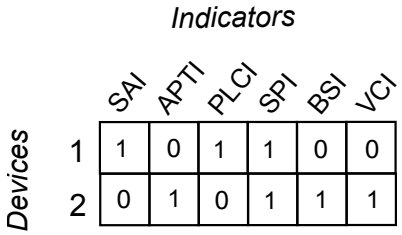


Figure 9.2 – An example of default objective weightings assigned to each indicator category per device.

For example in Figure 9.2, device number 1 is the PLCBM device, which has no access nor computational facility to calculate the BSI, business metrics nor evaluate the VCI, video capture of its own behaviour. The weighting for information on those metrics is thus zero.

9.2.5 Periodic Indicator Updates

Each device can use the up-to-date pre-evaluated performance indicator measurements as an input into their local classifier, as described below under detection modelling. In theory this enables an expert specified performance measurements of the changes to the system, be those changes repair or update. This also offers multi-layered decision making at a localised viewpoint, by using that measurement knowledge from other sources. This is essential to make the best decision, i.e. quickest with most up-to-date information. To do so, the detection classifier model must incorporate new input data with the pre-evaluated indicator knowledge.

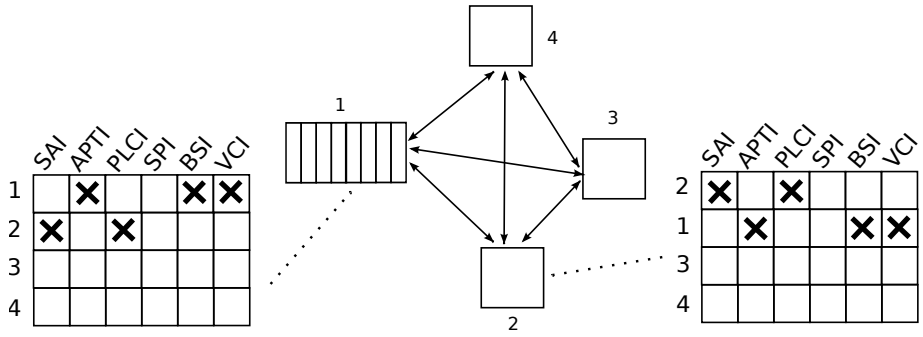


Figure 9.3 – An example of the current indicator states as viewed from each device, updated by periodic state message transmissions. X'es are values unavailable to that device.

The cost for this knowledge is in the data quantity of up-to-date measurement transmissions and enables localised decision making similar to the innate immune system. The alternative centralised, i.e. client and one server, or hierarchical, i.e. client and multiple servers such as our role separation approaches lead to a delay in decision making. In these cases a request for a decision is sent and result returned. Similarly to adaptive immunity using antigen presenting cells, the process pipeline is sense, send, schedule, decide, return and act. Bottlenecks will occur in scheduling at the server and on the network paths in transmissions with this latter approach.

A further challenge for the detection model and training the model is deriving semantic-time-relevance from the indicators, particularly in the business success indicators (BSI). For example, the BSI data will report the output state of the automation system. This implies the measure is an evaluation of automation activity already completed. From the perspective of other indicators, such as the real-time video capture indicator (VCI), the corresponding BSI indicator's relevant value is at a time in the future.

9.2.6 Current Trust and Self-Centric View

The *current trust* in received indicator measurements can be drawn by an individual node from the age of those measurements or whether an indicator is unexpectedly missing. This will cause each device to have a marginally different perceived system performance. Discovering whether deviations or anomalies, at this level, can be evaluated to indicate damage remains as an open research question.

The *ego-centric view* is to place the device's own indicator measurements above other devices' measurements. The matrices in Figure 9.3 show the order of indicator rows sorted by age. The first row is always its own row of data. The lower the row, the lower the priority of that information.

9.2.7 Fitness Function of Performance

The performance indicator data can be fused and evaluated in any number of ways to give a system performance or danger evaluation score. The example fitness function shown in Equation 9.1 takes account of initial belief weighting ($w_{i,j}$), current trust (t_j), a ego-centric view (w_j) and the present value ($V_{i,j}$) of the indicator from the device's matrix of indicator values. m and n are rows and columns of the matrices shown in Figure 9.3.

$$\sum_{j=1}^{m}\sum_{i=1}^{n} V_{i,j} * w_{i,j} * t_j * w_j \tag{9.1}$$

We arbitrarily limit the ego-centric view's effect by reducing linearly the weighting for each matrix row as: $w_j = 1.0, .., 0.75$. $V_{i,j}$ should be a positive MIN-to-MAX (from poorest to best) indicator measurement value. t_j should be evaluated using a error distance from normal learnt transmission delays. $w_{i,j}$ are assigned as described in 9.2.4, based upon the information accessible to each of the devices. Further consideration of data fusion techniques may be beneficial before implementation.

9.2.8 Data Transmission

There are three forms of data transmission, the high volume one-way transmission of traffic data from lightweight to intensive role devices for learning, the low volume one-way learnt classifiers and recovery solutions from the intensive to lightweight devices and the low volume two-way transmission of evaluated performance indicator states sent between all lightweight role devices.

The first enables learning the detection classifier models and recovery models. For this, the performance indicator data is required from within each PLC controller rack, from within the human machine interface (HMI) workstations and historian servers, etc. on the industrial network. These items of data are transmitted from these lightweight role devices to the intensive role devices with a relatively high frequency. The detection classifier model and recovery models are then returned periodically at a size much smaller than the original sent data. Thirdly, in order to make up-to-date detection decisions with network-wide knowledge summaries (performance indicator scores) we must transmit this information in a two-way decentralised manner.

To avoid a self-imposed denial of service scenario, we can employ engineered methods to ensure this data is transmitted in an intelligent manner. These include: data compression, selective data item transmission (which data to send), selective data item destinations (where to send) and selective periodic transmissions (when to send).

9.2.9 Detection Modelling

The detection modelling consists of data collection, accumulation and labelling, off-line data learning to generate a detection model classifier and redistribution of that classifier to all lightweight role devices.

The detection model will use a hierarchical classifier where each layer corresponds to an indicator category. The classifier will enable one of its layers' output to be exported and used as an input into the next layer of the same classifier in another device. This feature will permit fast decision making by transferring a partial pre-evaluated decision to each device. Each layer will output a numerical measure of danger corresponding to its indicators. These will be accumulated to give an absolute level of danger based upon each new set of inputs.

Feature selection is used to let the feature representation be led by the data rather than the designer. Data quality, such as information gain, is a typical measure for feature preference and is data-driven to intelligently direct the selected monitored data by analysing the collected data over time. Employing feature representation learning via feature selection results in two important outcomes, less data needed

for approximately equivalent accuracy. This means the traffic data collected and sent is the most indicative of normal or improved behaviour. Thus this can be used to reduce the amount of and focus on the most informative data collected, sent for learning and input into the classifier.

Periodic re-evaluation of the classifier model will enable regular monitoring of the dynamic normal behaviour. Updating of the detection mechanism is in part analogous to an evolving self-healing action. The detection modelling will receive input traffic and sensor data which is labelled with the performance indicator evaluation scores. The task at the intensive role device(s) will be to uncompress and accumulate the received data in time-series. The data will be time-correlated to the performance indicator fitness scores, either using some aggregate or average over all devices or the ego-centric viewed evaluation score from the sender of the specific item of data.

The intensive feature learning and classifier modelling will occur anywhere within the trusted distributed system as a task for the intensive role. The ideal location is to employing dedicated parallel processing hardware, i.e. a computing cluster, and software, i.e. Apache Spark (Zaharia *et al.* , 2010) and Hadoop (White, 2009), to distribute the off-line data mining task. In this work we do not propose the details of the model learning nor the output classifier, only the required theoretical information semantics as specified within this text.

During an initial bootstrap period overseen by human operators a set of performance indicator fitness scores will be recorded, these will provide the benchmark for good system performance. The indicator scores are required as numerical from MIN-to-MAX values, reflecting a corresponding worse to better performance.

Changes to the automation system via upgrades or replacements will affect the performance indicators and the system evaluation fitness score, thus the data labelling and the detection modelling will be impacted. To workaround this case, we resort to human operator white-listing via a user interface. A future solution will be to use the system evaluation fitness score, as a true measure of performance, to determine whether the change was beneficial and therefore revoke poorer performing changes. We believe the performance fitness score will also hold the key to generated recovery solution validation for future self-healing security systems.

9.2.10 Extracting Recovery State Models

The recovery creation module will extract from monitored Supervisory Control and Data Acquisition (SCADA) network protocols transmission packet logs upload binaries and parameter write data to the PLCs, permitting automated restore to previous configurations. This static recover approach enables us to take advantage of the enduring decision by Siemens (on many of their S7 PLCs) and the International Organisation for Standards (ISO) to not encrypt automation instruction commands at these layers. The packets can be collected when received at the PLCBM device and when sent from an HMI or other device. In the binary upload case, we can extract the payloads from a TCP/TSAP steam, recombine the byte data as the original binary and attach the new binary to another S7 Comms transmission at will. In the parameter write case, we can simply extract the parameter value and its target data block (DB) address.

The extracted and time-series stacked artefacts of parameter values and a checkpoint of binaries will form the recovery solutions. Once extracted the intensive role will distribute the relevant known good

parameter change stack and the binary checkpoint stack. An S7 comms protocol transmission script, i.e. SNAP7/MOKA7 (Nardella, 2013), and ordered argument list is enclosed in the recovery tool. The script will execute the recovery solution and will confirm that the PLC's block data timestamps and the parameter values are as expected. In the event of a failure to verify the change an alert can be issued to the expert operators and the next argument on the stack can be attempted. Further failures can either be terminated with a STOP_PLC command sent and an alert issued.

By storing the most relevant recovery solutions on the PLCBM the update can be issued immediately. The decision to execute the recovery is either after an alert to and approval by a human operator or after a timeout in cases of a higher success probability. Probability of recovery success can theoretically be achieved by reinforcement of earlier user approval to similar contextual conditions and performance indicator scores to those earlier cases.

9.3 Self-Management Component

Further adaptive and intelligent mechanisms will add additional security, autonomy and redundancy to the distributed self-healing security system. We include these components in this work for their consideration. A collective awareness algorithm can monitor other devices to improve the trust assessment and better localise malicious activity in the network. A moving target strategy can enable obfuscation of the security system's behaviour profile to avoid or hide itself from malicious attacks. Dynamic recovery generation tools can find and evaluate new repairs to revert or improve the system performance. Regional node clustering and bottleneck avoidance can be employed to optimise application layer payload routing and focus the repairs to a localised region. Each has its own computational costs, challenges and application specific requirements and restrictions to manage in order to be applied to industrial networks.

9.3.1 Social Sensing and Collective Awareness – *"the guards themselves become the threat"*

In the innate and adaptive immunity cellular-membrane levels and in social animal groups, the members will observe the behaviours of other neighbouring individuals to evaluate the degree of trust in the other individual. This is often referred to as *social sensing* in academic literature. The trust level is communicated to other members and used for praise, i.e. increased belief in data from that device, or segregation, i.e. phagocytosis or exclusion from the group.

In Figure 9.4 we illustrate an example of social sensing in an industrial network. In each intensive role (IR) device a map of normal behaviour is built per neighbouring device. A behavioural map of device (A) is collected at two or more IR devices and enables a collective decision to be made when a change in device (A)'s behaviour is noticed.

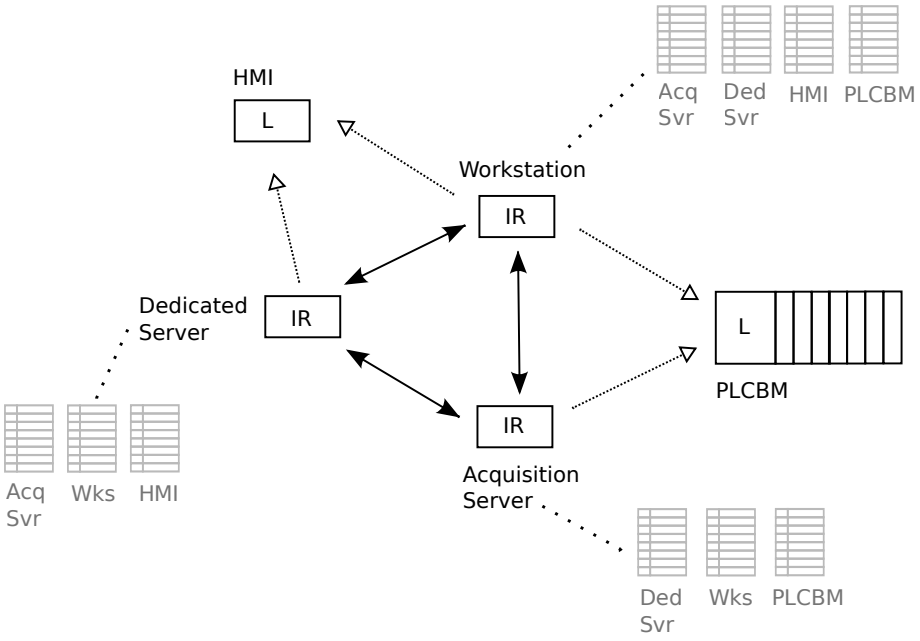


Figure 9.4 – Diagram of intrusive social sensing behaviours as depicted by one-way requests (white arrows) and two-way requests (black arrows). Grey tables at intensive role (IR) devices show the collected local perspective knowledge of the other devices' normal behaviours.

We have selected a simple intrusive request mechanism inspired by (Brumley *et al.* , 2007)'s locally evaluated self-verifiable antibody-alert (SVAA) mechanism, see 4.2.8. A node will receive and run a request command, then return the result. The approach gives an unrestricted range of possible data collection request tasks, however it is not without limitation. There is a weakness of returning falsified values if the execution environment is not trusted, and care should be taken to ensure the request command does not damage the recipient node.

The issuance of a request command can be implemented using a conventional information request mechanism such as remote procedure calls (RPCs) or an application programming interface (API). The interface should enable receipt of two forms of generated commands. Firstly built-in commands and secondly a formulaic question with a range of possible variables; e.g. "frequency of x per interval of y seconds" or "frequency of x in automation project directory y". For example, the frequency of a 'specific statement list (STL) or ladder logic (LAD) function code' in automation project 'Proj1' or the frequency of 'network data bytes received/ sent' per interval of 'two' seconds.

The intrusive requests should aim to return behaviour or status information. A sensing mechanism should then be able to discern normal from non-normal behaviour or be able to correlate to worsening performance indicator scores. Expert-directed industrial network security metrics should be considered, the two metrics drawn above are adapted from (Knapp, 2011, p196).

The non-normal recognition and decision to act is carried on three devices (A,B,C) as follows:

(1) Device (B) recognises an anomaly in its collected data from device (A).
(2) (B) will ask all trusted devices (i.e. C) with (A) data whether (A) is acting anomalously.
(3) (C) will respond with its assessment score.
(4) (B) will combine the scores, reach a decision and act.

The collective awareness mechanism enabled by social sensing allow us to finely monitor device

behaviours from a group perspective and will lead to improved trust assessment and better localisation of non-normal or malicious activity. The mechanism uses and contributes to the performance indicator evaluations and trust beliefs within the self-healing system component. This concept and area is presently under-explored in existing literature, in the context of decentralised or swarm-based security systems. The minor drawback to this approach is a imposed increase in computation and local storage requirements. The more critical concern is the increase in the quantity of two-way network transmissions at potentially high volumes to enable up-to-date multivariate monitoring.

9.3.2 Moving Target Strategies – *"who will guard the guards themselves"*

Future self-healing security systems will need to handle attack mitigation activities as a response to a recognised attack, in addition to the collaborative repair and immunisation actions of self-healing. In (Crouse & Fulp, 2011) and (Fink & Oehmen, 2012) the authors address moving target avoidance strategies by using a genetic algorithm to evolve less vulnerable configurations for the host operating system and its network-facing applications. In those works the systems change from vulnerable to less vulnerable.

Another problem exists where a device may be a victim of distributed denial of service (DDoS) on a specific port number or where a router is reconfigured to drop packets of a specific protocol or where a process is targeted and its heap space becomes corrupted, in essence *survivability*. In these and similar cases, collective swarming behaviour via a change in configuration, such as spawning and disowning a fresh process or initiating a coordinated change in communicating port number or protocol, is a necessary mechanism for self-healing in our context. (Polack, 2010) discusses this in the context of highly connected information systems, whereas (Owens *et al.* , 2007) offers a theoretical and generic architecture toward building these homeostatic immunity behaviours, at this time neither have citations in applied computer security. Blocking mechanisms (Kang *et al.* , 2011) and time-to-live blacklisting signatures (Swimmer, 2006) have been used to avoid or mitigate the effects of DDoS attacks. However, where the information system is affected by an uncategorised threat, i.e. as indicated by a decaying performance evaluation score, the execution of coordinated behavioural changes upon the distributed system remains under-explored.

9.4 Strategy Generation and Evaluation: Static vs. Dynamic

Using static strategies in moving target and recovery generation is inadequate as a solution when the reliant systems change or when targeted by intelligent and autonomous malware. While our static recovery approach may be translated to automation systems of other manufacturers that use the ISO-TSAP protocol or by direct cooperation with the PLC manufacturer, we don't see it as a final solution. When the ISO-TSAP or S7 protocol standards or implementations change the recover extraction strategy will lose its value.

A dynamic viewpoint on repair generation is likely to be the path of fruition. Other authors have noted that code can be validated within a virtual sandbox environment (Brumley *et al.* , 2007), (Newsome *et al.* , 2006), (Newsome *et al.* , 2005) and that evolutionary algorithms can be used to generate and evaluate those repairs using unit testing (Le Goues *et al.* , 2012) and runtime crashes

(Perkins *et al.* , 2009). Ensuring absolute correctness in modelled industrial network environments in which a recovery solution will execute is challenging. PLC simulation environments exist to assist code development and testing normal and failure cases, such as sensor and mechanical failures. However automating the validation of controller code under extreme external changes remains a challenge. If it can be done, an evaluation score of the automation system behaviour and the system-wide performance indicators would be suitable to direct the automated repair generation. Then for each solution an evaluation score can be calculated by executing the solution within the model. Over an evolutionary period the best solution can be evaluated, selected and approved for use. Application of these dynamic generation approaches is more flexible to dynamic changes in industrial networks and will thereby facilitate future self-healing security systems.

9.5 Application Focus

The application goal of this architecture is upon automated assembly lines and life-critical automation processes such as helicopter pan-tilt stick control, among other networked real-time and automation systems.

9.6 Challenges

Several challenges of the system have been identified. Implementation and validation of the architecture for application to industrial networks remains open. Two other unavoidable problems are managing the computational expense on self-healing systems on networks of heterogeneous nodes and large quantities of data transmissions required by decentralised distributed systems operating on real-time information with many sources.

Managing the computational expense of learning data from multiple sources is an infrastructural challenge addressed in this theoretical architecture. The presented solution separates and offloads intensive processing tasks from industrial network devices with low capability or availability to other devices with higher availability. This enables classifiers to be trained and delivered to the low capability devices.

Large data transmission quantities are unavoidable in peer-to-peer topologies operating on real-time distributed data. In our architecture these occur in the quantity of one-way transmitted traffic required for learning, the two-way state data updates and the two-way social sensing state data requests. Questions on reduction of data transmission quantities remain open for future work. Possible approaches include data transmission heuristics specific to decentralised systems; such as using predictions of other node's data transmission content and their destinations to help improve information quality per transmission. An open question exists over whether the data transmissions (identified above) will remain at an acceptable throughput-level to maintain the real-time performance of the self-healing application and the automation operations.

9.7 Chapter Conclusions

This chapter described the CARDINAL-E theoretical decentralised self-healing system architecture for application to industrial networks. The detection and recovery solutions are applicable to automation networks that use the unencrypted ISO-TSAP SCADA network protocol for binary uploads and parameter writes to the controllers. The design has been inspired by the cellular interaction and organisational behaviours of the biological immune system and is stated as a self-healing component and a self-managing component. The path ahead will be to evaluate the feasibility, safety concerns and correctness of the presented features and components in testbed lab trials before moving onto industrial networks.

Chapter 10

Conclusions

This thesis has made advances in both applied and theoretical self-healing computer security systems for industrial automation networks, including those supporting critical national infrastructure. Based on this research into academic artificial immunity, an evaluation framework and a theoretical architecture for self-healing and collaborative security systems have been designed, presented and undergone prototype testing for application to real-world Industrial Control System (ICS) and Supervisory Control and Data Acquisition (SCADA) networks. Specifically a foundational piece of this theoretical architecture, *CARDINAL-Vanilla* an immunity-inspired decentralised security module distribution platform, has been formally defined in Chapter 5 and rigorously assessed on virtual and real-world enterprise networks in Chapters 7 and 8 using our new evaluation framework introduced in Chapter 6.

The key hypothesis of this work was defined in section 1.5, it stated: "In the context of a distributed and self-healing system, compared to engineered approaches a reactive multi-agent and holistic immune system inspired approach will have better distribution, and thus capability for self-healing, performance over a range of networked environments. Where performance is measured by a 'self-healing immunisation rate', consisting of the number of items transmitted, time to distribute an item and data sent to distribute those items, in order to assess the objectives of self-healing and application feasibility to industrial networks."

Chapter 7 conducted a comparative analysis of the prioritisation and distribution mechanisms of an equivalent engineered algorithm, called *Optimal-Static*, to the *CARDINAL-Vanilla* architecture's algorithm. The evaluation was performed under virtual and enterprise network types with a multi-objective metric to measure their applicability to networks of certain throughput capacities. In section 7.7 the key hypothesis was rejected in all tested cases for low-throughput networks (O2), such as industrial networks, and it was rejected in all tested cases, except in networks of ten nodes, for applicability to high-throughput networks (O3), such as enterprise networks. The rejections were confined to the stated test design methods and test parameter configurations.

The research questions stated with the key hypothesis in section 1.5, were: "To what extent will a decentralised self-healing security system inspired by an holistic view of the biological immune system, be able to distribute network transmitted security modules (i.e. detectors and repair tools) in a network of heterogeneous devices – such as an industrial control network? How will its self-healing and network transmission performance compare against an engineered system."

The amount by which *CARDINAL-Vanilla*'s results were poorer under the enterprise network tests

reported in section 7.4 is not relevant to a real life application. Those enterprise network results showed comparable *Distributed-M1* and *Time-M2* metric performances to the engineered selection and distribution algorithm. *CARDINAL-Vanilla* sent more data which led to an overall poorer immunisation rate performance. The virtual network tests in section 7.2 reported poorer *Distributed-M1* distribution performance as the network size increased by comparison to the real networked tests. The further analysis in section 7.6 indicated that the real network tests should be used for interpretation as they removed ambiguity from the results.

In section 7.7.1 we argued that Optimal-Static's algorithmic approach is inappropriate to implement in a real system as it assumes perfect conditions and is thus prone to poor performance under failures, see section 7.2.2. To remove this assumption from the algorithm requires further data transmissions which will in turn reduce its measurement scores.

Chapter 8 presented a sensitivity analysis of *CARDINAL-Vanilla*'s key parameters. Section 8.3.3.4 indicated that an independent change to parameters $P0, P2$ and $P3$ of the default *CARDINAL-Vanilla* parameter configuration would lead to a statistically significantly improved result set measured under the low-throughput networks multi-objective metric (O2), as compared to the configuration evaluated in Chapter 7. Under the high-throughput networks multi-objective metric (O3), section 8.3.3.4 indicated a significant and improving effect will result by an independent change to parameters $P1$ and $P3$. Section 8.7.2 provided a small set of parameter configurations, without the $P4$ parameter as it reported an insignificant effect. Comparing these configurations to the engineered approach remains as open work.

10.1 Contributions for Industrial Network Security

We have proposed and reported on Distributed Self-Healing Security Systems (DSHSS), a new perspective advocating intelligent, host-based collaborative support and learning as an infrastructural security architecture, to address the computer security challenges affecting the ICS and SCADA networks problem domain specified in Chapter 2. Our view sits amongst a growing handful of existing theoretical academic works, as specified in Chapter 9.

It is our belief that nature inspired algorithms will provide robustness of a range of conditions, and we persist to believe they can be applied to maintaining automation process up-time via automated self-healing detection and recovery and via infrastructural architectures. Our self-healing system benchmark experimentation of *CARDINAL-Vanilla* in Chapter 7 has shown that the resource expense for detection and distribution is certainly feasible for application to the corporate network devices of industrial networks, where small quantities of data are learnt on every device.

For real-world automation systems the future work architecture in Chapter 9 offers a mechanism for monitoring and optimising automation process instructions via a reinforcement methodology. It offers performance indicator and recovery mechanisms to establish a causal relationship leading to better or worse performance. This approach can be applied to cyber security applications and to automation process benchmarking and improvement as the system operates over its lifetime under different instruction configurations. Today instructions are written by engineers, tomorrow using the proposed future architecture, instruction improvements can be automated and guided using this approach. This sets a roadmap for future DSHSS operating on industrial networks.

The evaluation methodology in Chapter 6 enables assessment of DSHSS, including distributed collaborative security systems and collaborative Host-based Intrusion Detection Systems (HIDS). The systems performance measures assess self-healing capability and resource feasibility for application suitability to enterprise networks and to ICS and SCADA control networks. Classification accuracy and self-healing quality metrics remain to be included. These items will need to be integrated into the methodology in cooperation with the future work architecture. A number of other future work items are mentioned at the end of each chapter.

10.2 Contributions for Artificial Immune Systems

The thesis holds several items of interest to the Artificial Immune System (AIS) research community. We have introduced an expanded theoretical standpoint on self-healing, self-organisation and other self-* behaviours for computer security applications as led by our biological theory analysis work in Chapter 3 and surveys of artificial immunity in Chapter 4. With this new perspective we have developed the *CARDINAL-Vanilla* architecture in Chapter 5 and our future work architecture in Chapter 9 that embodies those concepts and self-* principles in a real-world applicable manner to address security problems in industrial control system security.

We have extended existing AIS work based on (Kim *et al.* , 2005)'s abstract danger theory-inspired *CARDINAL* model to combat malicious spamming software. Our work has adapted its use case to a distributed defence application similar to collaborative HIDS. For the first time *CARDINAL*'s abstract model has undergone formal mathematical specification in Chapter 5 and key differences between our *fair* implementation of Kim et al.'s model called '*CARDINAL-Vanilla*' and Kim et al.'s original model have been specified. Thorough parameter range evaluations under several rigorous testing conditions have been undertaken in Chapter 8. Rigorous comparison evaluations of *CARDINAL-Vanilla* against an optimal engineered system were conducted in Chapter 7. This latter work reported similar self-healing performance scores between the two systems, however the bio-inspired distribution mechanism sent more data. Recommendations to modify the distribution heuristic from commonality- and importance-based to another or combined measures were discussed within to rectify this difference.

The work into AIS modelling and metaphorical *immuno-engineering* and the testing in Chapter 8 has led us to show our experience of common AIS algorithm components, such as decay, suppression, proliferation and focus when in danger in section 8.2 and conclusions in 8.8. Interpretation of these items carries a warning, of consideration only with respect to their function within the mathematical multi-agent *CARDINAL-Vanilla* model expressed in Chapter 5 and with respect to the testing conditions. A key lesson learnt was that *CARDINAL-Vanilla* has a wide range of parameter configurations that give good and similar performance scores in section 8.8. This owes to its robustness over parameter configuration changes, a feature, we expect, may be common among communicative multi-agent-based systems.

10.3 Summaries and Conclusions

In Chapter 2 we investigated the causes of software security vulnerabilities and architectural problems faced by industrial automation networks that are in operation today. Through this investigation we

identified the necessity for collaborative and automated self-healing security software, a view that we share with other experts in the field. The chapter concluded with the uncovered main architectural challenges affecting the security of industrial automation networks.

In Chapter 3 we began to address the self-healing problem in industrial networks by expanding on recent findings and theories in biological immunology for their cellular component and interaction behavioural traits that lend themselves to self-healing in the human body. We drew out the key characteristics and *immuno-engineering* mechanisms that lead to biological self-healing and homeostatic behaviours in human immunity. The chapter concluded with a mapping from the main challenges to the biologically engineered solutions.

In Chapter 4 we followed up this line of enquiry in existing artificial immunity. Here we explored the bio-modelling and bio-inspired metaphorical approaches taken by authors in the AIS field. Following this we surveyed research works on distributed self-healing architectures that employed immune system correlations to the detection and response against malicious software attacks. This work led us to select the abstract *CARDINAL* architecture model by (Kim *et al.* , 2005) as an underlying design for the platform.

In Chapter 5 we formalised the mathematical model of the *CARDINAL* architecture, extended and described its implementation-level definition into *CARDINAL-Vanilla* using a distributed multi-agent system representation based upon immunological danger theory. Stemming from the agent interactions and their constraints, we determined the key emergent behaviour of *CARDINAL-Vanilla* were led by the priority and distribution heuristic and the voluminosity-based dispatch decisions. With the belief that these modular decisions had the most influence upon the local decentralised decision making and upon the speed of immunisation, we prepared an evaluation methodology in order to test these components and *CARDINAL-Vanilla*'s self-healing capability.

In Chapter 6 we explained the principles behind the evaluation and validation for our *CARDINAL-Vanilla* platform, and those that also apply to other DSHSS. These principles addressed user behaviour modelling, globally measured metrics to assess network-wide performance, multi-objective optimisation evaluations to draw simple meaning from the metrics, data sources, configurations to enable real-time read-in of data and a two-phase experimentation method to evaluate self-healing, among others. This evaluation methodology was experimentally benchmarked in Chapters 7 and 8, has shown to be reliable on conventional and virtualised networks, and is applicable to future DSHSS on industrial networks.

Because of the sensitive nature of the real-time operations on industrial automation networks and the academic novelty of incorporating a non-engineered decision system into this environment, it was necessary to thoroughly assess *CARDINAL-Vanilla* in real computer networks. In the first benchmarks the evaluation methodology and the architecture were tested for their stability under virtual network and real network conditions. The platform was compared against equivalent engineered solutions in Chapter 7. The results showed that *CARDINAL-Vanilla* had similar *immunisation rate* performance on real-networks to our engineered solution. However, it transmitted comparatively more data on larger sized networks, which is due to its data-driven selection mechanism of distributed security modules. We pinpointed the algorithm's distribution heuristic as the key component that will improve the system's module and destination selection and thus lower the quantity of data it transmits for distribution.

In Chapter 8 we assessed the architecture under ranges of parameter configurations to find the set that reported best improvement in our immunisation rate system performance objectives. This

enabled a thorough benchmark of the architecture's underlying AIS model, its algorithms and its configurations. Two parameter tuning searches reported a number of good performing configurations under virtual and real networks. The results showed that some regions within the parameter ranges degrade the immunisation rate performance scores to an extreme. However, many parameter space regions exist where performance is broadly good and similar. Therefore we found that the platform and AIS model show remarkable robustness over many parameter configuration ranges when applied to our problem domain.

In Chapter 9 we presented a future work infrastructural DSHSS architecture that aims to address the problem cases of decentralised and automated self-healing systems for industrial networks. The design is guided by immunity principles in self-healing and self-management. It incorporates recovery mechanisms using SCADA network protocols, damage detection via performance indicators and real-time decision making using network-wide contextual information by dispatching partially evaluated decision scores. This provides a future roadmap for security information infrastructural architectures and Distributed Self-Healing Security Systems (DSHSS) in industrial networks.

Appendix A

Security in Industrial Networks

A.1 Industrial Security Standards

A.1.1 Organisations with Released Industrial Security Standards

The following are names and acronyms of the publishing organisations of Industrial Control System (ICS) standards, and other acronyms. Note ANSI/ISA and ISO/IEC are collaborators on their released standards.

- National Institute of Standards and Technology (NIST)
- North American Electric Reliability Corporation (NERC)
- International Electro-technical Commission (IEC)
- International Organization for Standardisation (ISO)
- American National Standards Institute (ANSI)
- International Society of Automation (ISA)
- Critical Infrastructure Protection (CIP)
- Critical National Infrastructure (CNI)

A.1.2 Released Industrial Security Standards

Table A.1 lists the current standards designed toward securing ICS and electronic networks.

Organisation	Standard	Purpose
ISO/IEC	27001:2005, 27001:2013	Standards for information security management.
ISO/IEC	27002:2005, 27002:2013	Code of practice for information security management controls.
NERC	CIP002-011, Revision 5	Standards for bulk electric systems and network security.
NIST	800-82, Revision 2	Guidelines for ICS security –performance, reliability, safety requirements
ANSI/ISA	62443 , ISA-99	Comprehensive standards and technical reports for electronic security of ICS.

Table A.1 – Standards for securing ICS and electronic networks. Organisation acronyms are listed in A.1.1.

A.2 Vulnerabilities Reported on OSVDB 2007–2015

The following table contains the raw quantity data from searches within the OSVDB (Kouns & Martin, 2015) database of vulnerabilities. The OSVDB database is formally and consistently tagged by dedicated staff, sponsored by Risk Based Security in Richmond Vancouver, Canada.

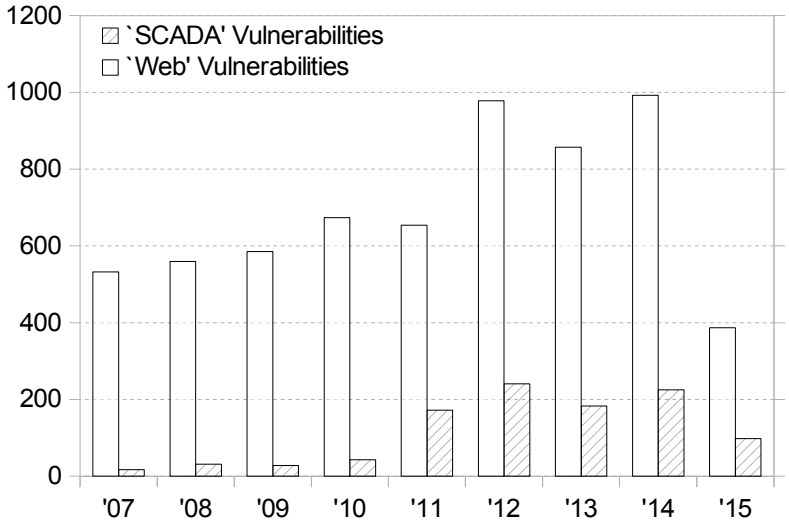
A.2.1 SCADA and Web Vulnerabilities Reported on OSVDB

The search terms used are quoted within the table headers.

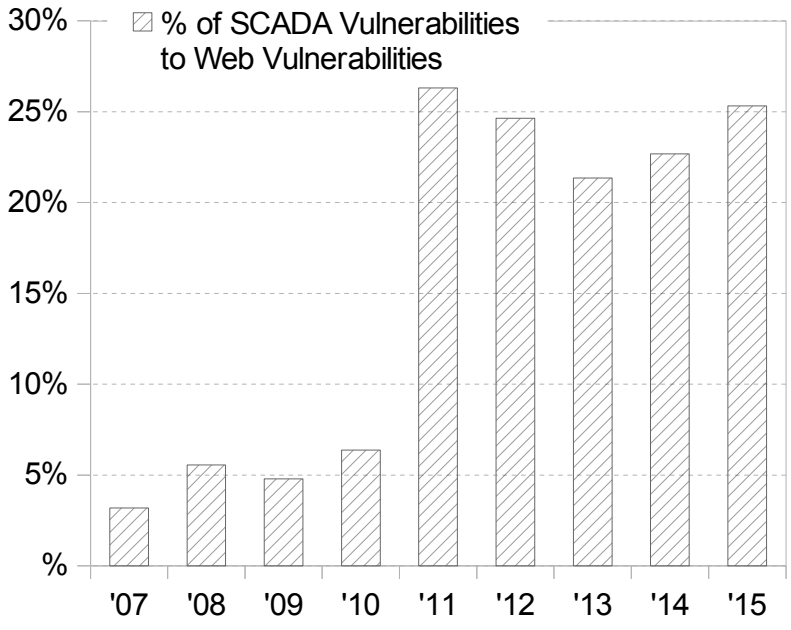
Year	'SCADA' Vulnerabilities	'Web' Vulnerabilities	% of SCADA Vulnerabilities to Web Vulnerabilities
'15	98	387	25.323%
'14	225	992	22.681%
'13	183	857	21.354%
'12	241	978	24.642%
'11	172	654	26.300%
'10	43	674	6.380%
'09	28	585	4.786%
'08	31	559	5.546%
'07	17	532	3.195%

Table A.2 – Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.*

We compare 'SCADA' to 'Web' vulnerabilities as the latter are perhaps the most commonly discussed and reported of exploits.



(a) Quantity of reported vulnerabilities.



(b) % of vulnerabilities tagged "SCADA" vs. "web".

Figure A.1 – Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.2.

A.2.2 Vulnerabilities per PLC Manufacturer Reported on OSVDB

The search terms used are quoted within the table headers and prepended to the keyword "SCADA".

Year	'Siemens'	'Schneider' Electric	General Electric ('GE')	'Rockwell' (Allen-Bradley)	'Honeywell'	'Emerson'	'ABB'
'15	22	9	3	4	1	2	1
'14	36	22	2	4	26	4	0
'13	36	19	12	6	2	4	3
'12	37	33	28	12	1	7	2
'11	24	12	6	5	2	0	1
'10	1	1	0	8	0	0	0
'09	0	0	0	0	0	0	0
'08	2	2	4	6	0	0	1
'07	2	0	3	0	0	0	0
Total	160	98	58	45	32	17	8

Table A.3 – Vulnerabilities per PLC manufacturer, ordered by total reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.*

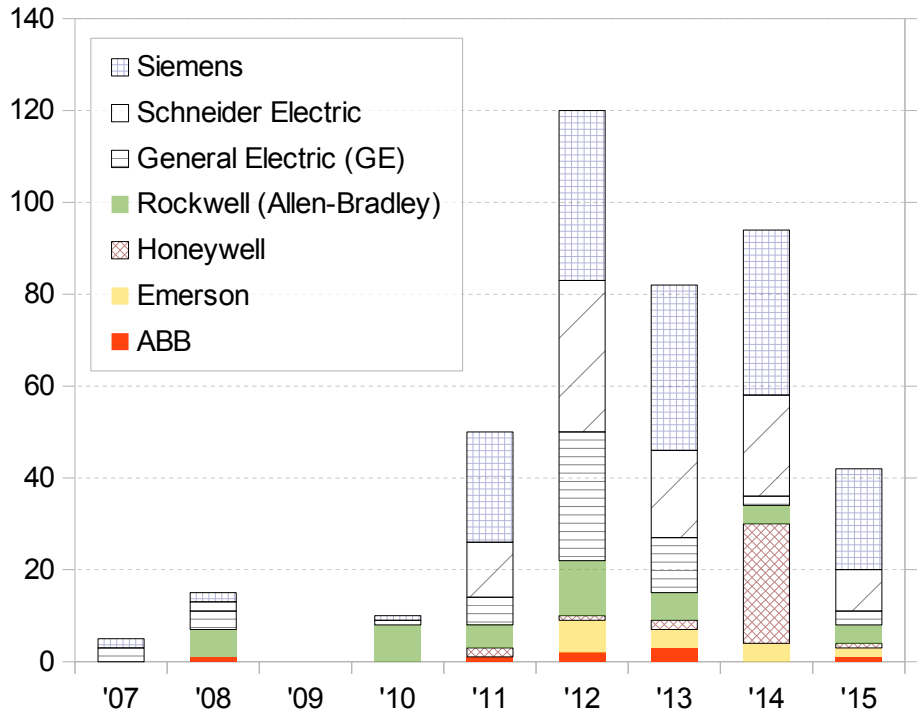


Figure A.2 – Vulnerabilities per PLC manufacturer, ordered by total reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.3.

A.3 SNAP7 - Open Source S7 Communications API

Figure A.3 shows a compatibility table of controller functions (first column) remotely called via the S7 networked communications protocol for the specified Siemens controllers (top row). The open source SNAP7 application programming interface (API) facilitates the manufacture and transmission of these packets to Siemens programmable logic controllers (PLC).

Table available at http://snap7.sourceforge.net/snap7_client.html *Retrieved on 2015.08.20.*

S7 Protocol partial compatibility list (See also § LOGO and S7200)

	CPU						CP	DRIVE
	300	400	WinAC	Snap7S	1200	1500	343/443/IE	SINAMICS
DB Read/Write	O	O	O	O	O	O(3)	-	O
EB Read/Write	O	O	O	O	O	O	-	O
AB Read/Write	O	O	O	O	O	O	-	O
MK Read/Write	O	O	O	O	O	O	-	-
TM Read/Write	O	O	O	O	-	-	-	-
CT Read/Write	O	O	O	O	-	-	-	-
Read SZL	O	O	O	O	O	O	O	O
Multi Read/Write	O	O	O	O	O	O	-	O
Directory	O	O	O	O	-	-	O	(2)
Date and Time	O	O	O	O	-	-	-	O
Control Run/Stop	O	O	O	O	-	-	(1)	O
Security	O	O	O	O	-	-	-	-
Block Upload/Down/Delete	O	O	O	-	-	-	O	O

Snap7S = Snap7Server

(1) After the "Stop" command, the connection is lost, Stop/Run CPU sequence is needed.
(2) Tough DB are present and accessible, directory shows only SDBs.
(3) See S71200/1500 notes.

Figure A.3 – Table shows compatibility of controller functions (first column) remotely called on the specified Siemens controllers (top row) as invoked via the Siemens S7 networked communication protocol.

A.4 Extract of ICS Attack - Water.arff Dataset

Below is an extract from the water.arff dataset released in May 2013 as part of the ICS Attack dataset by Mississippi State University. The dataset is known to be unsuitable for machine learning as "attacks were perform[ed] (SIC) with [an attribute set] to one value (x) and normal operation with another value (y)", thus making its class separability unrealistic (Adhikari *et al.* , 2013). Note also the limited quantity of network packet or stream data attributes.

```
@attribute 'command_address' real
@attribute 'response_address' real
@attribute 'command_memory' real
@attribute 'response_memory' real
@attribute 'command_memory_count' real
@attribute 'response_memory_count' real
@attribute 'comm_read_function' real
@attribute 'comm_write_fun' real
@attribute 'resp_read_fun' real
@attribute 'resp_write_fun' real
@attribute 'sub_function' real
@attribute 'command_length' real
@attribute 'resp_length' real
@attribute 'HH' real
@attribute 'H' real
@attribute 'L' real
@attribute 'LL' real
@attribute 'control_mode' real
@attribute 'control_scheme' real
@attribute 'pump' real
@attribute 'crc_rate' real
@attribute 'measurement' real
@attribute 'time'  real
@attribute 'result' {'0', '1', '2', '3', '4', '5', '6', '7'}

@data

7,7,183,233,9,10,3,10,3,10,0,25,21,90,80,20,10,2,1,0,1,85.7589569091797,1.00,0
7,7,183,233,9,10,3,10,3,10,0,25,21,90,80,20,10,2,1,0,1,85.6736755371094,1.07,0
7,7,183,233,9,10,3,10,3,10,0,25,21,90,80,20,10,2,1,0,1,85.616828918457,1.16,0
7,7,183,233,9,10,3,10,3,10,0,25,21,90,80,20,10,2,1,0,1,85.5599746704102,1.10,0
7,7,183,233,9,10,3,10,3,10,0,25,21,90,80,20,10,2,1,0,1,85.4747009277344,1.15,0
```

Appendix B

Biological Immune System

Figure B.1 summarises cell and protein signalling behaviours within the human body.

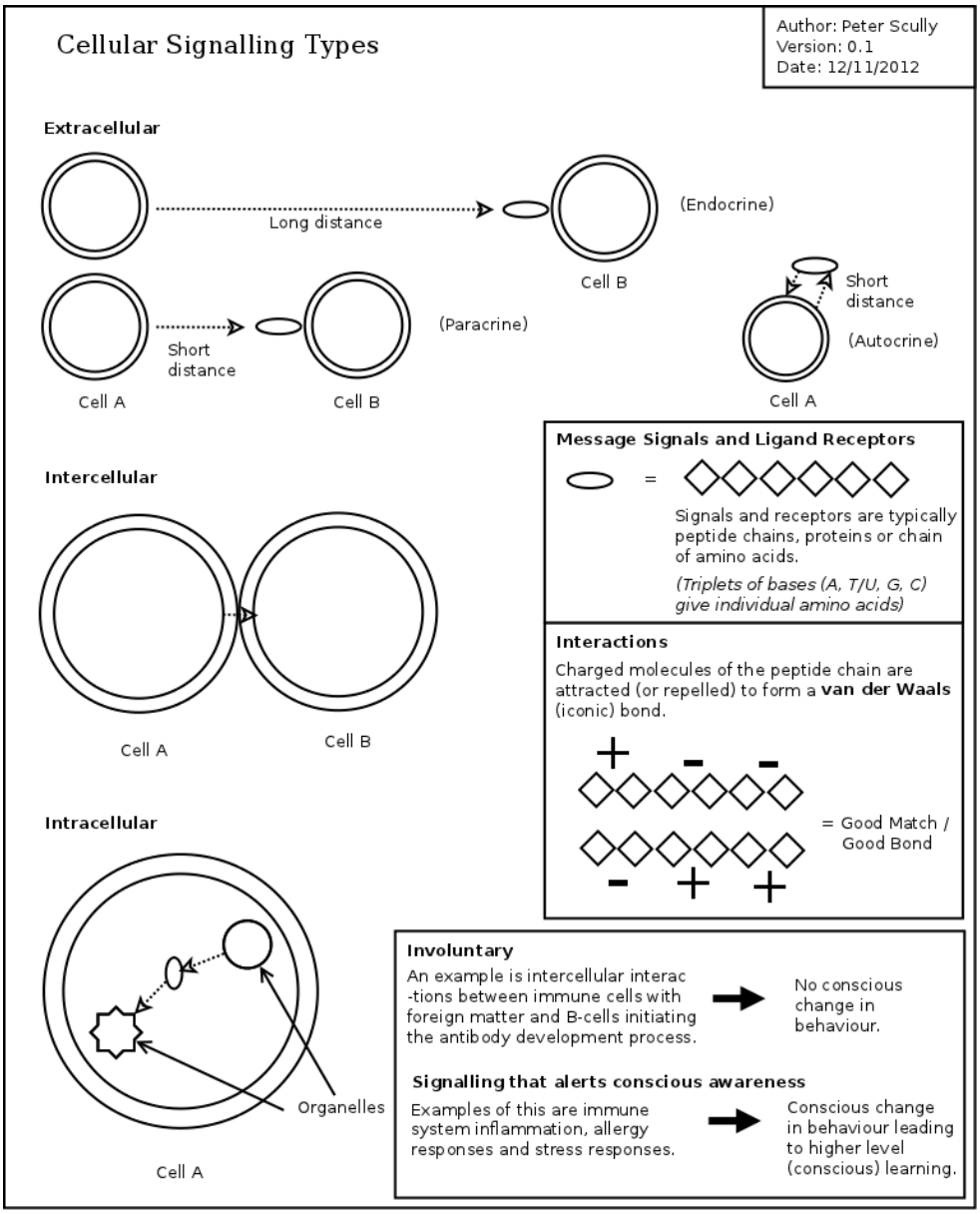


Figure B.1 – Cellular signalling types.

Appendix C

CARDINAL-Vanilla Architecture

C.1 Implementation Specific Responses

To block incoming protocol-port network traffic from the sender, i.e.:

```
iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -j DROP && sudo /sbin/iptables-save
```

To send an email to inform administrative staff of an imminent threat:

```
sendemail -f fromuser@gmail.com -t touser@domain.com -u subject -m "Type II danger
signal received from 10.10.10.10" -s smtp.gmail.com -o tls=yes -xu gmailaccount
-xp gmailpassword
```

To issue a STOP_PLC command to terminate operation on the microcontroller. This can be done on Siemens SIMATIC S7 series programmable logic controllers (PLC) in the following way using the opensource SNAP7 or MOKA7 library[1] (Nardella, 2013), as:

```
S7Client Client = new S7Client();
Client.SetConnectionType(S7.OP);
Client.ConnectTo(IpAddress, Rack, Slot);
Client.PlcStop();
```

[1]SNAP7 library download and documentation address: http://snap7.sourceforge.net/

C.2 Use of Random Number Generators

The implementation of CARDINAL-Vanilla makes use of three instances of the linear congruential generator (LCG) algorithm to produce random numbers. In each case we use Java's `java.util.Random` class with a pre-determined 48 bit seed. LCGs are known to be prone to produce a lack of *randomness* in their number generation, in fact Java's LCG (RAND) algorithm implementation has been shown to score worst out of a selection of random number generator (RNG) in the *diehard test* in (Meysenburg & Foster, 1999); see (Marsaglia, 2003) for an explanation and update of the original *Diehard Battery of Tests for Randomness*.

The pitfalls of LCGs are well known, in particular sequences of the lower order bits perform poorly on the diehard RNG testing suite. As shown in (Marsaglia, 1968) and more recently in (Gärtner, 2000) and (Coffey, 2013), sequences of those lower order bits are generated in planes and are thus particularly inappropriate in geometric (i.e. 3 to n-dimensional) applications when used as coordinates. Of course the advantages of LCGs are that they're written into most standard language libraries, they're fast and they're deterministic.

An Analysis of java.util.Random

Our tested implementation uses Java's Random method `nextInt()`. The method produces 32 bit two's complement big-endian integers. Figure C.1 shows a graphical reiteration of the point made above, in that we notice the darker (bit index with value mostly equal to 0) and lighter (value mostly equal to 1) bits are mostly at opposite ends (of the binary representation). In Table C.1 we show the mean frequency counts of bits equal to 0 (black in the figure) of each of those bit indexes for 16384 (128^2 calls to the method). The mean bit values presented are calculated over 32 separate seeds (0-31). These empirical tests using seeds 0 to 31 show very low standard deviation in Table C.2, relative to the range of possible values; except in the case of the signed (first) bit index. There is also a tendency that in each byte, bit index position 4 (from left) carries a higher probability to be zero (is darker). An inferential significance analysis of correlation between numbers selected by differing seeds remains as open work.

In conclusion, the descriptive statistics (and in part the cited references) strongly suggest that Java's `nextInt()` method using LCG will more probably result in smaller numbers (within its possible range of 0 to 32^2 and maximum range of 0 to 48^2) rather than larger numbers (in the range). We find this to be the case given the default seed and within those seeds presented in 0-31 range (used by up to the first 32 nodes of the self-healing benchmark experiments) when given the default value limit and over 128^2 generated random numbers. However, as the quantities of lines in our datasets are within this lower region, we do not see this as a pitfall in our case.

16384	8203.1	8174.3	4096.9	8197.6	4104.5	4084.4	2043.9
8198.3	4107.2	4080.8	2043.3	4102.7	2055.1	2038.8	1016.4
8202.4	4104.5	4093.1	2048.3	4101.8	2053.1	2044.2	1021.2
4109.4	2055.2	2050.3	1022.9	2056.3	1028.2	1024.8	508.3

Table C.1 – Tabular representation of related Figure C.1, showing mean of frequency of 0 bit values (black) per bit index using seeds 0 to 31. Rows 1 to 4 show bits 0-7, 8-15, 16-23, 24-31. See text and figure description for details.

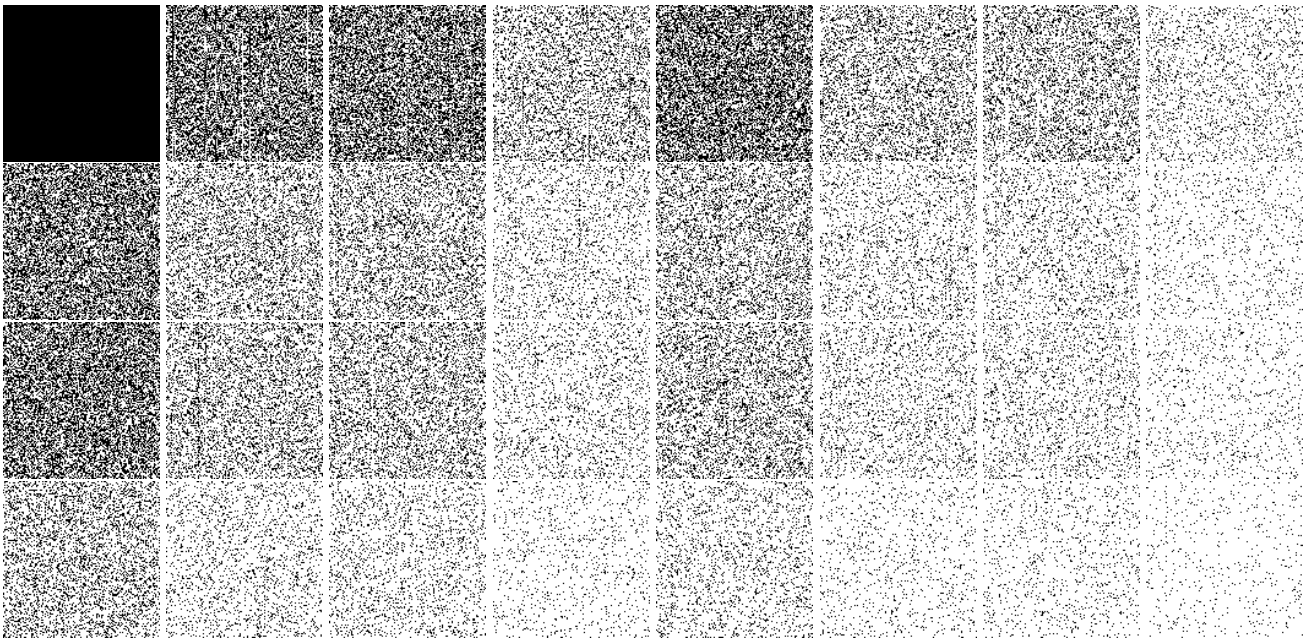


Figure C.1 – Figure showing 32 images produced by java.util.Random's use of linear congruential algorithm with default seed and limit on nextInt(). 1 image for each of the 32 bit values of integers, using two's complement (signed bit is index 0) with big endian representation (8th bit in 4th byte represents the smallest number). Black and white represent 0 and 1 (non-zero). Rows 1 to 4 shows bits 0-7, 8-15, 16-23 and 24-31. Images are scaled down from 128 by 128 pixels over 128^2 bits.

Bit	Std (freq.)	Bit	Std (freq.)
0	0.0	16	75.8
1	63.0	17	55.8
2	60.1	18	60.9
3	48.1	19	38.0
4	83.4	20	75.8
5	60.8	21	46.5
6	58.3	22	48.7
7	41.8	23	32.1
8	62.2	24	55.2
9	52.7	25	35.3
10	48.0	26	46.2
11	33.2	27	28.3
12	63.3	28	52.0
13	45.5	29	30.2
14	37.1	30	33.0
15	24.9	31	18.8

Table C.2 – Table showing standard deviation of frequency of 0 bit values (black) per bit index, using seeds 0 to 31.

Appendix D

Configurations

D.1 Virtual Machine Environment Configuration

Item	Configuration Value
Operating System	Fedora 16
Bash	4.2.37(1)-release (x86_64-redhat-linux-gnu)
Java	"1.7.0_09-icedtea"
OpenJDK Runtime Env	fedora-2.3.5.3.fc16-x86_64
OpenJDK 64-bit server VM	build 23.2-b09, mixed mode
Ulimit	ulimit -c unlimited
	ulimit -n 2000
	ulimit -u (9+(n*4))*n)+1024
Java arguments	-d64
	-Xms (16GiB-512MiB)/n
Swap	swapoff -a
Kernel cache	echo 3>/proc/sys/vm/drop_caches

D.2 Enterprise Machine Environment Configuration

Item	Configuration Value
Operating System	Linux v3.8.0-25 kernel operating system
Java	1.7
Ulimit	ulimit -c <default>
	ulimit -n <default>
	ulimit -u <default>
Java arguments	-d64
	-Xms 1024M
CPU	Intel Dual core E2220 2.4 Ghz
Memory	3,826 MiB
Network Adapter	Intel 82566DM-2, width: 32bit, clock: 33Mhz, 1Gbit, wired.
Network Switch	Dell PowerConnect 5448 switch, 1 Gbit with 48 ports.
- Port Config	1000-BaseT

D.3 Virtual Machine Environment and Execution Script

In this section we describe the environment and script configuration, section D.1 summarises the description.

The bash script, which executes the Java code, is run within GNU bash version 4.2.37(1)-release (x86_64-redhat-linux-gnu). The code is run with Java version "1.7.0_09-icedtea" using the OpenJDK runtime environment (fedora-2.3.5.3.fc16-x86_64) with OpenJDK 64-bit server VM (build 23.2-b09, mixed mode).

In order to successfully run, we must set the resource limit `(ulimit)` on the maximum size of core files created to `ulimit -c unlimited`, the number of open files allowed to `ulimit -n 2000` (for example opened sockets, streams to network destinations and file system inodes) and the maximum number of processes available to a single user to `ulimit -u (9+(n*4))*n)+1024`. Where the number of threads per *CARDINAL* instance is $9 + (n * 4)$). The Java virtual machine requires four, five threads are required to execute the *CARDINAL* model code and $n * 4$ threads are required for two sockets and two streams, one for both sending and receiving network communications. 1024 is the operating system's default maximum number of user processes, which we retain in case of other (unplanned) executing processes. $*n$ is to ensure that all instances can co-exist for the duration of the experiment, without any being blocked by the limit.

The Java code is run in 64bit mode with the argument -d64. We set an equal amount of allocated heap space memory per instance, with a maximum limit of 1024 megabytes. The minimum quantity is set to $(16GiB - 512MiB)/n$, such that for 10, 15 and 20 hosts $\{1024, 1024$ and $793.6\}$ megabytes were allocated. $512MiB$ was left for the Fedora 16 operating system to operate. To reduce unnecessary I/O waiting times debugging STDERR and STDOUT were disabled and experiment logging is buffered in memory and output to file at the end of each experiment trial run. This (eventually) leads to an exponential scaling factor on the memory footprint size (i.e. for transmission logs), but is encompassed by our large heap size allocation described. These relatively high values were precautionary allowances based on our assumption of memory required with some empirical trial and error. In future tests, this will need to be considered with respect to the dataset size and the maximum memory footprint.

The process of improving the reliability of the experiment lead to investigating the removal of operating system noise effects. The trialled actions that follow were not used during tests leading to the presented results as we found they had no noticeable influence on the reliability on the preliminary results. They are worth noting as they are likely to have an effect under similar test conditions. To avoid periodic disk to memory swapping we tried disabling the swap mount point, or `swapoff -a` under Linux. To avoid subsequent runs benefiting from memory cached file content we tried clearing the kernel cache between test trials, with `echo 3>/proc/sys/vm/drop_caches`. To reduce periodic processes starting we cleared the scheduled tasks (cron job) lists for the current and root users, with `crontab -e`. No further daemons or services were installed beyond the distribution's default set and we did not disable those running by default. We found the most commonly successful action for reducing the occurrence of intermittent execution failures were inspection of synchronized blocks and the immutability of data in the multi-threaded implementation; debugging at this implementation level can be a cause for much time investment, particularly in distributed applications that are running in an environment at the point of resource exhaustion.

D.4 Enterprise Network Environment Parameter Search Execution Script

The execution script consists of a server script, a client deployment script and a number of integrated modules. The server script begins by opening a remote terminal shell (Secure Shell (ssh)) on n machines, which downloads and executes the client script to run the experiment trials. The server script selects and passes arguments into the client script via the ssh session on each of the n machines. Each quantity of n has its own associated host configuration file(s), containing a pre-built connectivity mapping for

each machine. The server script loops through the quantity of parameter configurations c.

The client script performs the execution of the architecture algorithm for a number of trials where the parameter configuration file, host connectivity configuration file and experiment-specific configurations are parametrised. Upon completion of i trials the experiment logs on each machine are compressed and moved to a central location where they are labelled and parsed for their metric result values.

Termination of a single trial is coordinated by the architecture experiment messaging framework code and occurs (separately on the individual machines) when all machines have completed their role within the experiment procedure (see subsection 6.4.1). The next trial will begin on the minute after all n machines have reported the termination of their previous experiment run.

D.4.1 Time Evaluations of Other Datasets

The evaluation process leading to Table D.1 included model training within a single instance of the Weka data mining framework (Hall *et al.* , 2009). The reason for this test is to discover an approximate dataset parse time as given in columns Mean and Std. (sample standard deviation). We can then *approximately* estimate the memory and time costs required by CARDINAL to evaluate each dataset and estimate these costs as we increase the quantity of nodes.

The algorithm used was the pruned C4.5 decision tree (Quinlan, 1993). There is no doubt that the CARDINAL and C4.5 algorithms perform differently, have different actions and evaluations. Therefore, these values should be used as *approximate* estimates. The C4.5 algorithm was run with 0.25 pruning confidence threshold, 2 minimum instances per leaf, 2 fold error reduction, 1 fold pruning, seed value of 1 and uses multi-interval discretisation (Fayyad & Irani, 1993) to bin (group) continuous attributes. No other parameter conditions were evaluated. The parameters selected are defaults of the J48 implementation of C4.5 written by Eibe Frank (Hall *et al.* , 2009). The workstation used during the test was the same as that used in 7 such that a correlation of timings and the virtual benchmark tests should be solvable.

Dataset	**File**	**Time Taken** (sec.) **Mean**	***(Std.)***	**Size** (MiB)
CSIC-HTTP-2010	http-csic-2010-weka-with-duplications-utf8-escd-full.csv	573.5	(455.9)	97
KDDCup99-1999	kddcup.data-10-percent-corrected-ALL.arff	1542.5	(2.3)	53
gureKDDdataset-2008	gureKddcup6percent.arff	1051.4	(52.5)	31
ISCX-2012-9Flows-0%-to-10%-sample	TestbedThuJun17-2Flows.csv	10.4	(0.5)	21
ICSAttack-MississippiState-2011	water-final.arff	568.7	(9.6)	18
ISCX-2012-9Flows-0%-to-10%-sample	TestbedMonJun14Flows.csv	9.8	(0.4)	14
ISCX-2012-9Flows-0%-to-10%-sample	TestbedThuJun17-3Flows.csv	6.9	(0)	13
ICSAttack-MississippiState-2011	gas-final.arff	0	(0)	9.1
NSLKDD-2009	KDDTrain+-20Percent.arff	25.1	(0.3)	3.6
CSIC-HTTP-2010	sample-CSIC-2010-http-v0-5-2-sample-0p-to-9p-dataset	1.9	(0.3)	1.4

Table D.1 – Summary table showing dataset size and time taken to evaluate state of art network security SCADA and vulnerability security datasets using the C4.5 decision tree algorithm with one instance on one node. Time is taken over 10 runs.

Appendix E

Further Results and Analysis

E.1 Self-Healing Benchmark - Inferential Statistics

To expose the significance of the iteration independent results for both metric (M1,M2 and M3) and objective (O1,O2 and O3) results, we employ the following statistical tests and a null-hypothesis of equivalence between the AIS (Vanilla) and engineered (OptimalStatic) algorithms.

E.1.1 Difference from AIS to Engineered

Here we set the null-hypothesis as there is no significant difference, as follows:

Null Hypothesis 1. $H_0(A) : A_{IS} = A_{ENG}$

The immune system-inspired selection algorithm (A_{IS}) modelled in CARDINAL's implementation (CARDINAL-Vanilla) performs no differently than the engineered Optimal-Static selection algorithm (A_{ENG}) over each node network sizes. Using experiment procedure J with code versions J3 and J4 and evaluated according to the key measurement criteria with a number of repeated trial runs.

Theorem Specialisations

The key measurement criteria are M1,M2,M3 and O1,O2,O3 as have been discussed within the body of the thesis. Where M1,O1,O2 and O3 are measured with a preference on the highest value, while M2 and M3 are preferred with lowest values. The network sizes in the simulated (virtual network) tests are $n = [1, 5, 10, 15, 20]$ and $n = [1, 5, 10, 15, 20, 30$ and $41]$ in the enterprise network tests.

Which test and why?

The statistics test selection chart given in (Field & Hole, 2003)[p274] directs us to select the Mann-Whitney H non-parametric test (Mann & Whitney, 1947). In this case, the data are measurement scores from an experimental design using 1 independent variable (the changing algorithm), independent measures (the repeated iterations that do not affect each another) and with two groups (algorithms). We can safely assume the data is non-normally distributed, and thus the corresponding non-parametric Mann-Whitney U test can be selected.

The test ranks and produces its results using ranked data giving a U statistic; the value of inconsistent ranked sample values. Each test is carried out on each metric for each network size. The results upon the $H_0(A)$ can be found in Table 7.2. For example, the first test uses the input data of the metric M1 data of the four algorithm results with network size 1, which is identical for both algorithms, and thus shows as *invalid* (there is no significant difference).

Table description of the Mann-Whitney U and Cohen's d Tests

For the simulated trials, the results using the metric (M1,M2,M3) data are shown in Table 7.2, whereas the objective (O1,O2,O3) results are in Table 7.4 within its chapter. The enterprise network trial results are shown inTable 7.6 and Table 7.8 respectively.

Each table uses the same format, tabular structure and use the same test conditions. The first column (n) shows the number of nodes used during the given row of tests. Results are from a one-tailed Mann-Whitney test in the U statistic column with *df* showing degrees of freedom and the *P-value* is corrected to show a two-tailed significance value. The condition of $p =< \alpha$ is used to determine the significance column value as significant (sig.) or insignificant (insig.) where critical alpha is $\alpha = 0.05$. Cohen's d statistic function is defined in equation E.1, where σ and $\bar{x}$ are sample standard deviation and mean values. Its Effect Size categories are given in (Cohen, 1988)[p285], however our interpretation of Cohen's rating categories are expanded slightly to: *S (Small)* $> 0 <= .3$, *M (Medium)* $> .3 <= .5$, *L (Large)* $> .5 <= .7$, *VL (Very Large)* $> .7$. M_X refers to the mean value of the CARDINAL results and M_Y to the mean value of the OptimalStatic results for a given test.

$$d = \bar{x}_x - \bar{x}_y \Big/ \frac{\sigma_x^2 + \sigma_y^2}{2} \tag{E.1}$$

Note that reference median results can (and do) have small variations from zero on the objective evaluation results O1, O2 and O3. These evaluation measures have already been introduced in section 6.9 and the equation used to generate these scores was explained in subsection 6.10.3. These variations from zero are not unexpected behaviour. This is because the values are reported as medians of the ratio result of the equation, where the median reference value inputs (from M1, M2 and M3) into that equation are not necessarily from the same (single) instance. This can lead to the selected median reference instance for M1, for example, having a non-median representation in the objective evaluation space (e.g. O1) due to having a outlier as its M2 result. Upon return from the equation, the new O1 result for this example instance is distinctly non-zero.

E.2 Self-Healing Benchmark - Virtual Network Plot Results

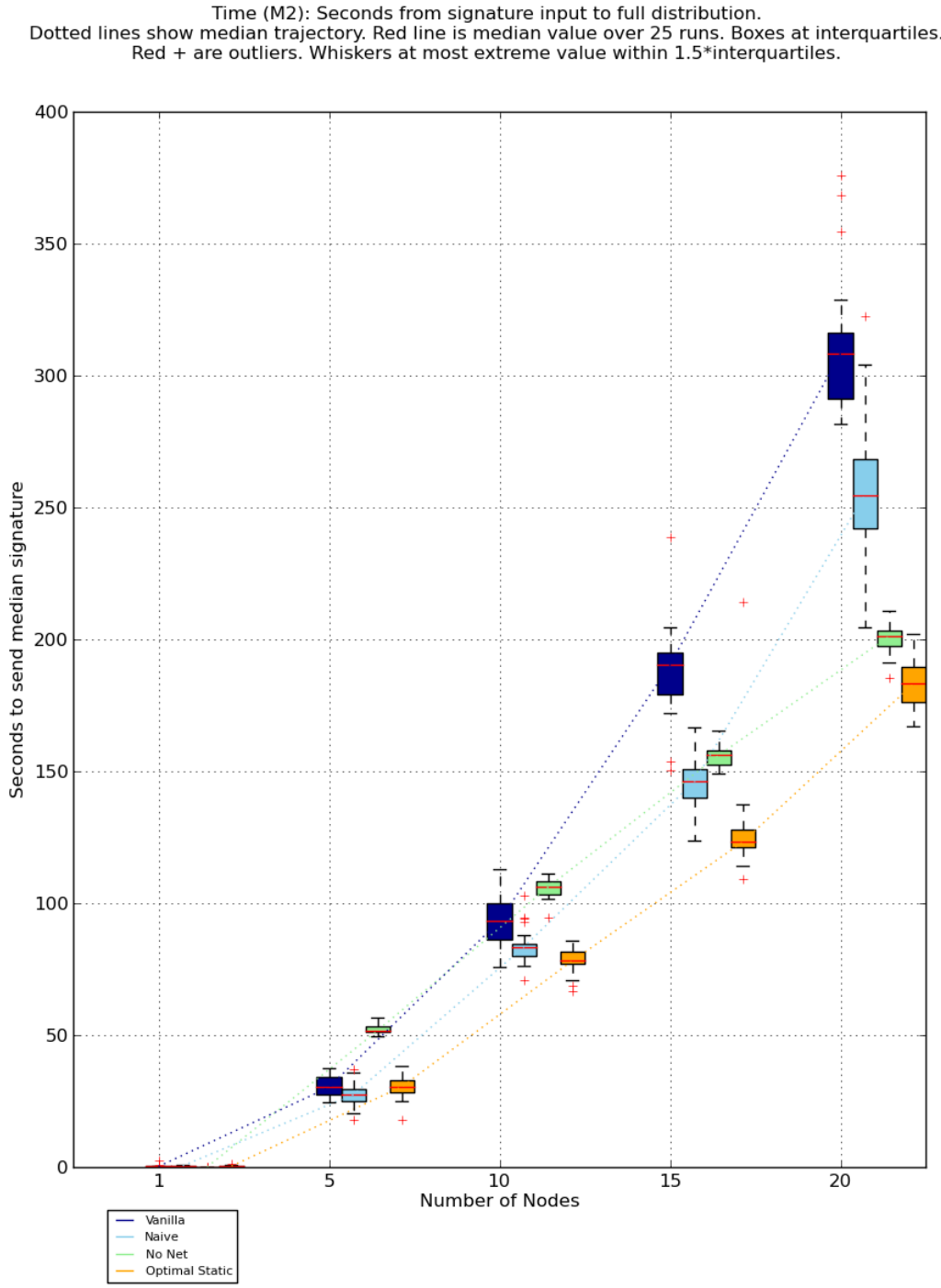
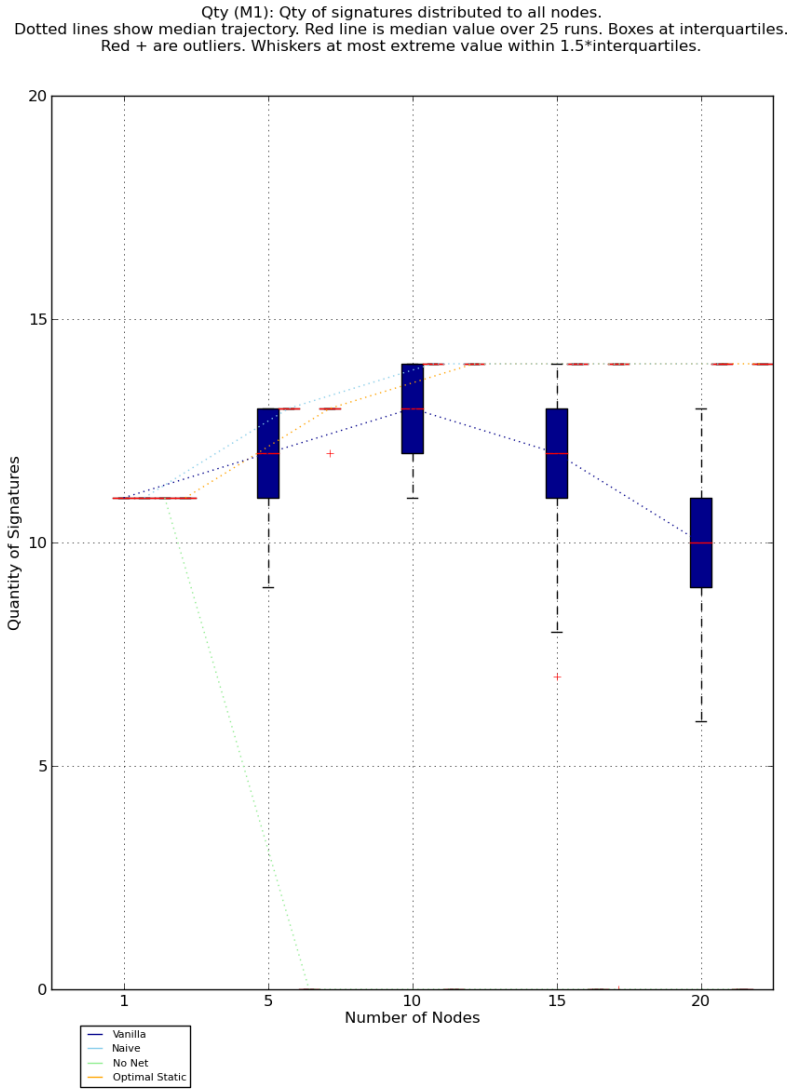
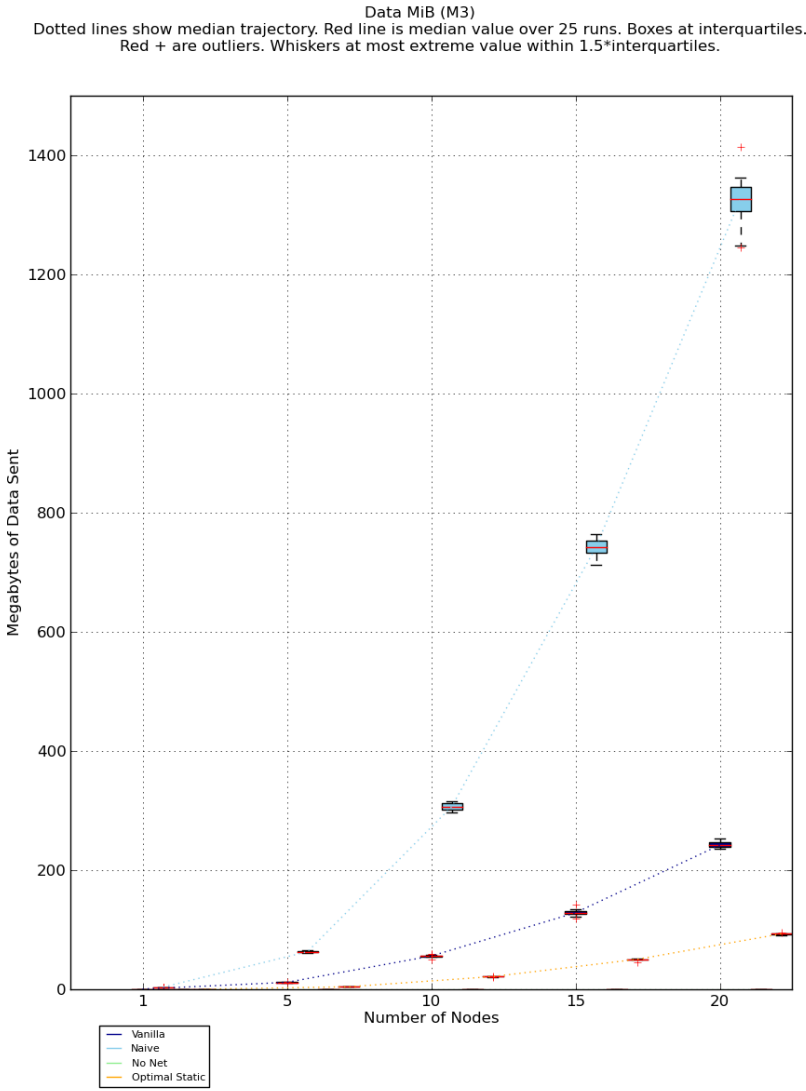


Figure E.1 – Time taken to transmit the median detector.

Note: A description of this plot is given in subsubsection 7.3.1.2 including discussion of how to interpret the misleading NoNet time results (green coloured boxes).

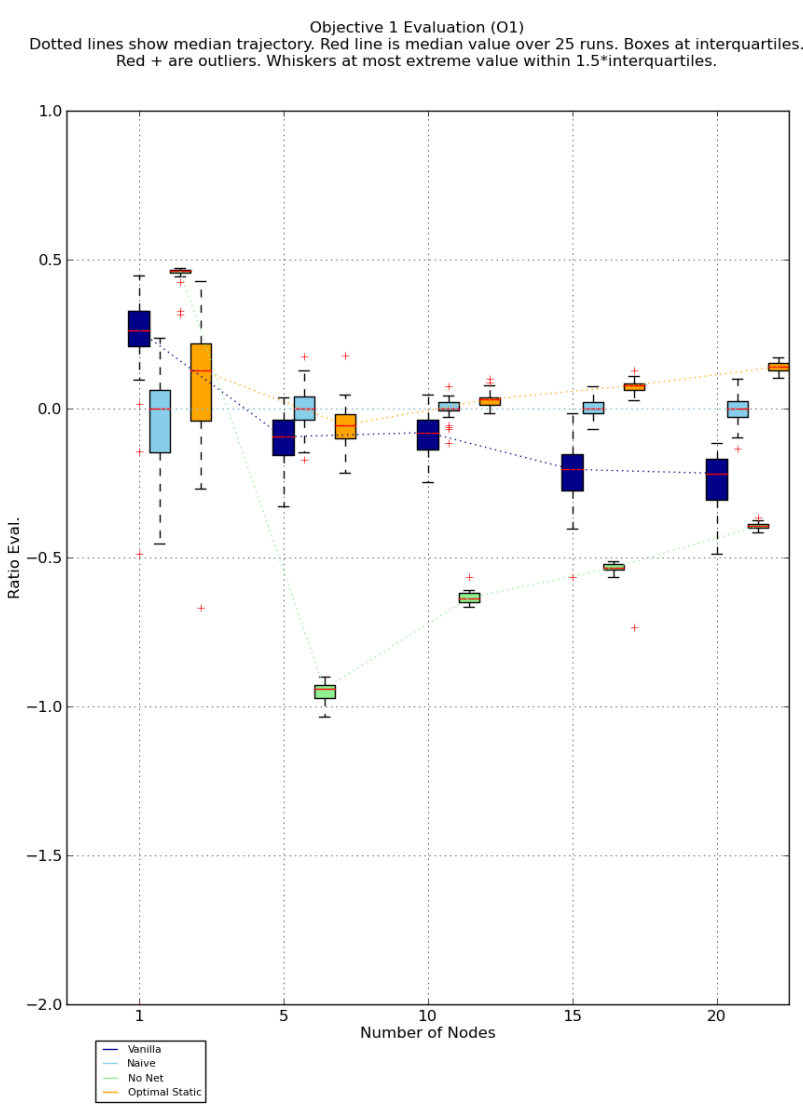


(a) Number of signatures distributed during experiment.

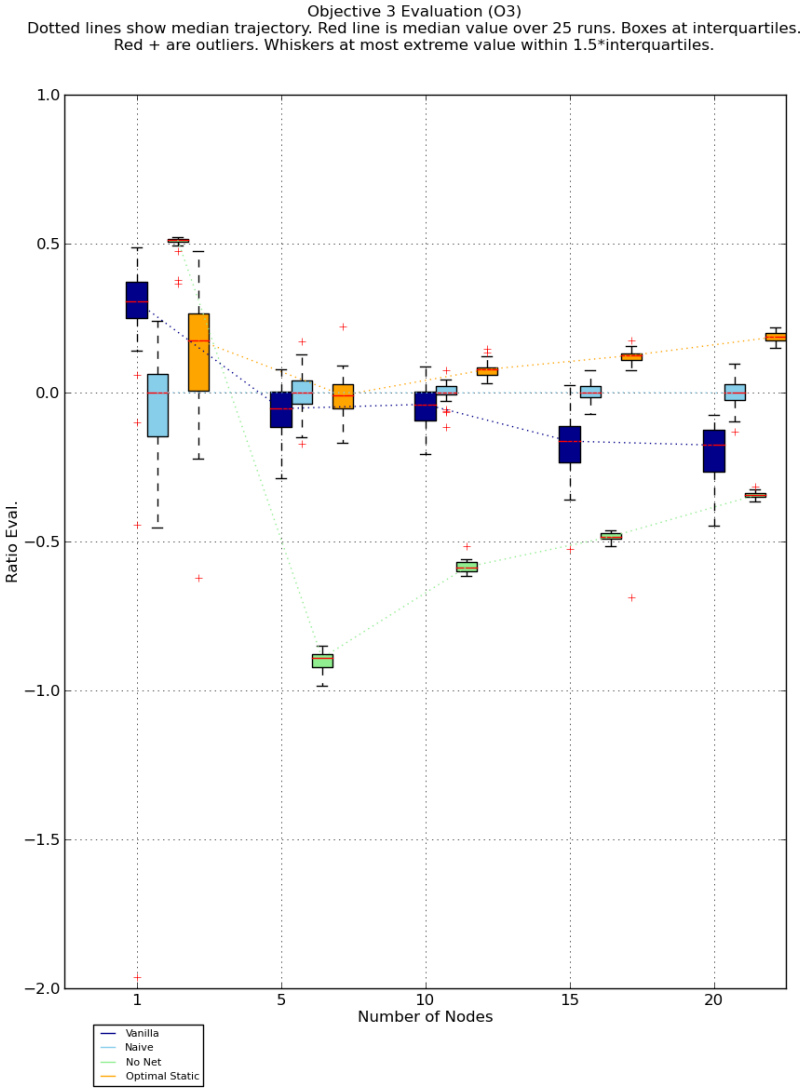


(b) Megabytes of data sent during experiment.

Figure E.2 – Box plots observations of metric results from the virtual network tests.



(a) Immunisation rate (O1), combining metric M1,M2 data



(b) Immunisation rate suitability upon enterprise networks (O3), combining metric M1,M2,M3 data

Figure E.3 – Box plot observations of our Self-Healing System research objective evaluations (O1 and O3) as tested on a virtual network. Using our extension of the Weisbin & Rodriguez ratio equation, with the SendAll algorithm results used as reference. Greater values imply better performance.

E.3 Self-Healing Benchmark - Enterprise Plot Results

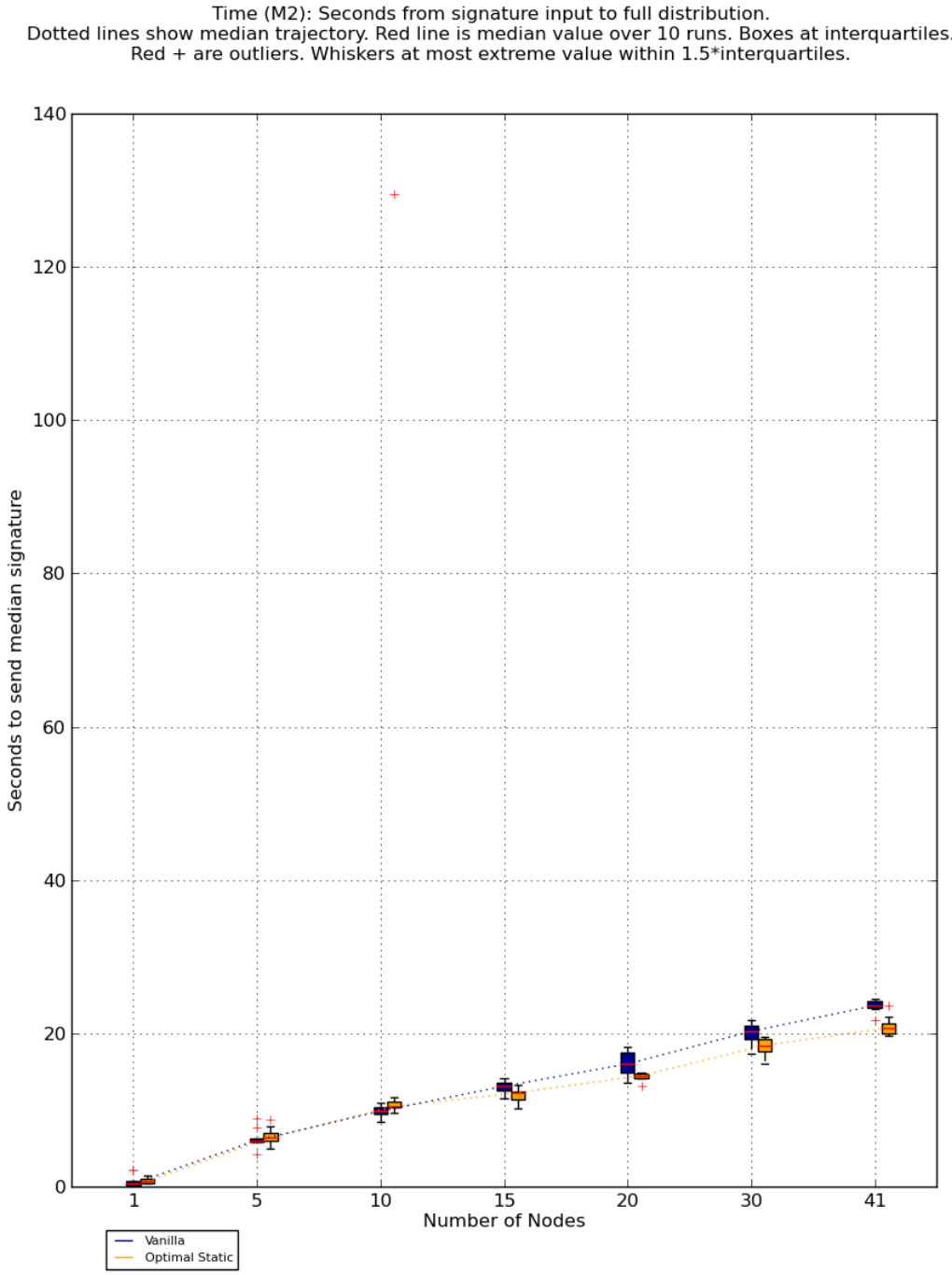
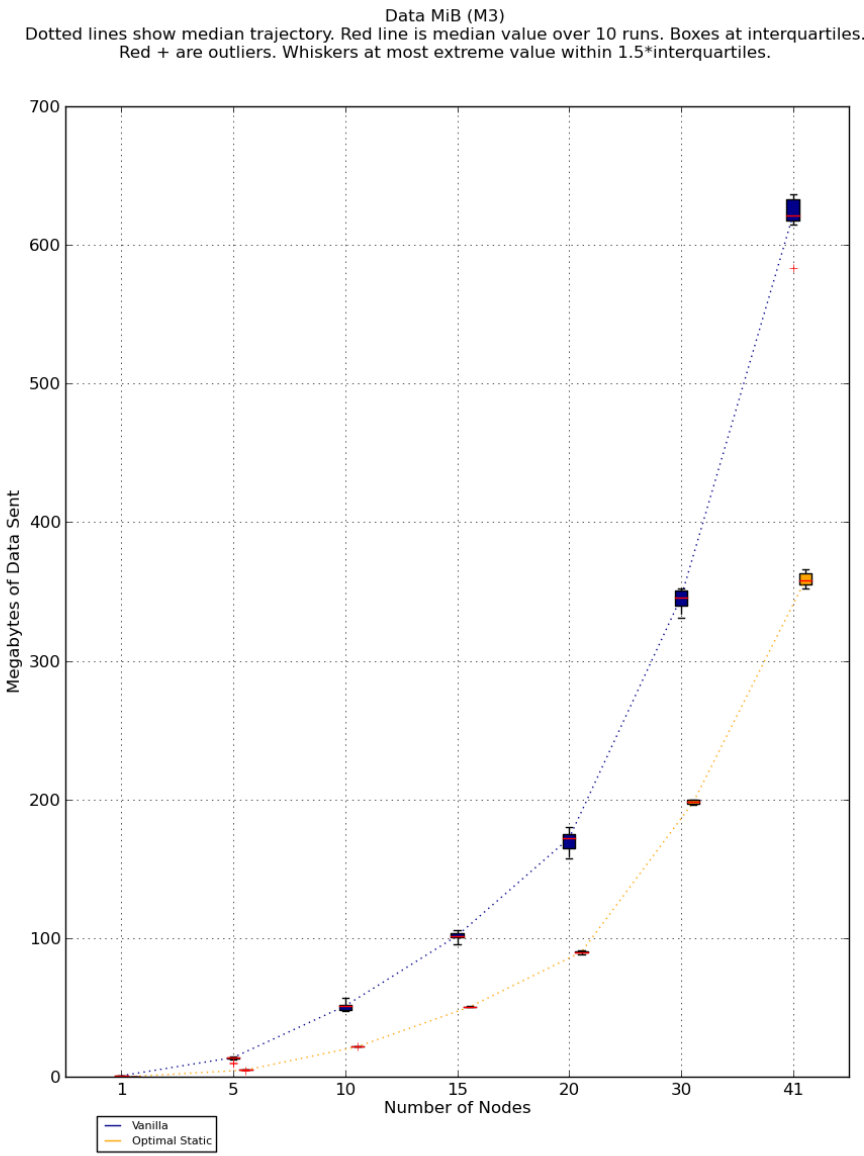
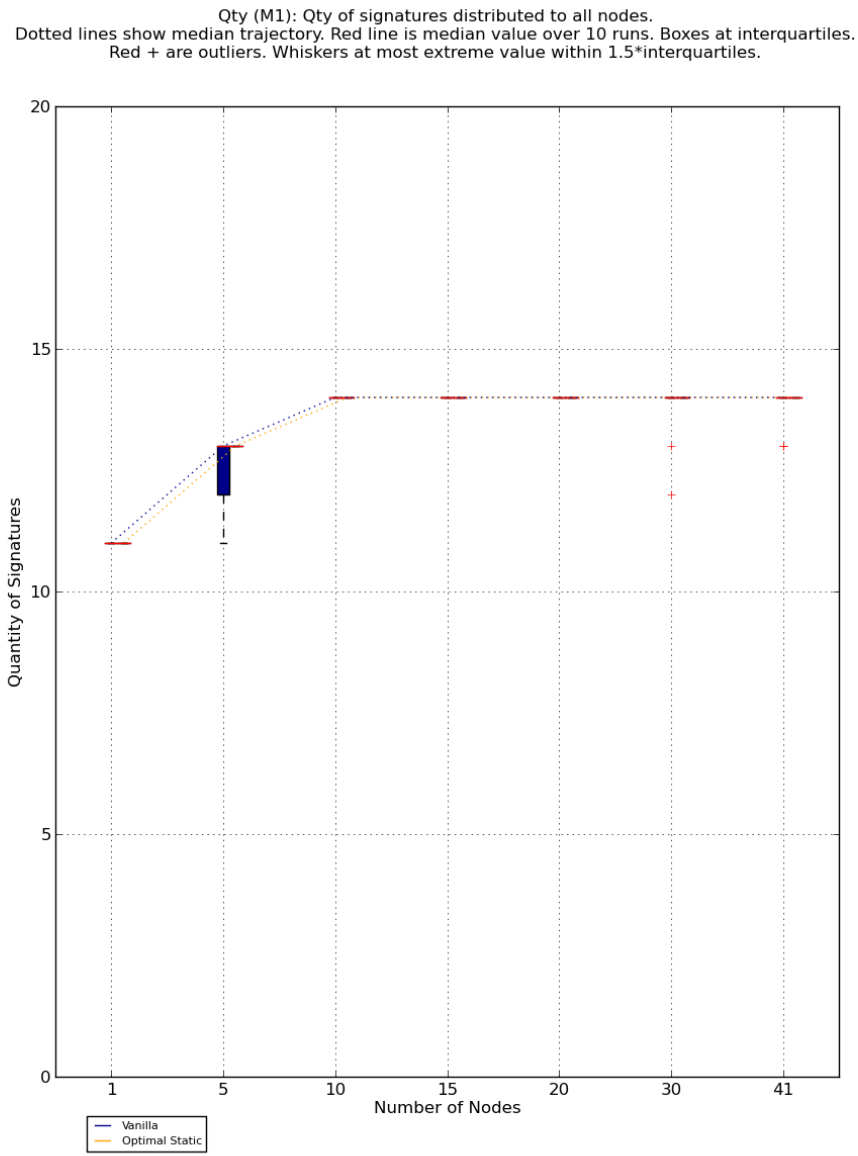


Figure E.4 – Time taken to transmit the median detector.

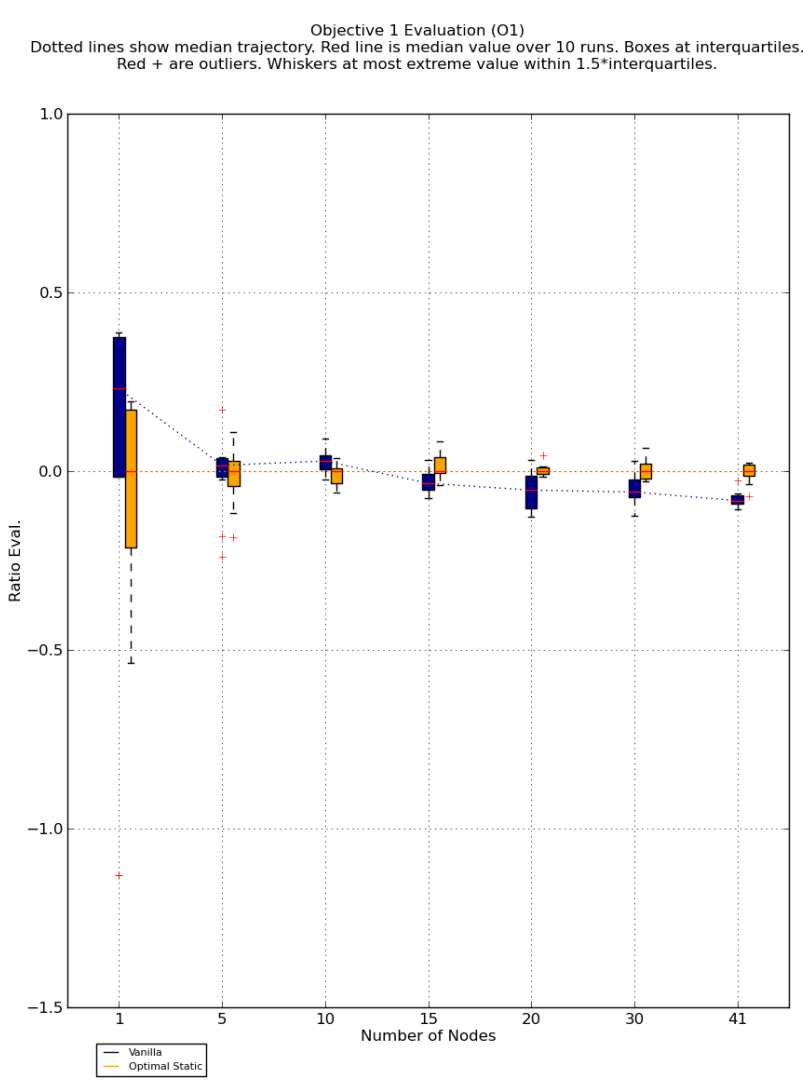


(a) Megabytes of data sent during experiment.

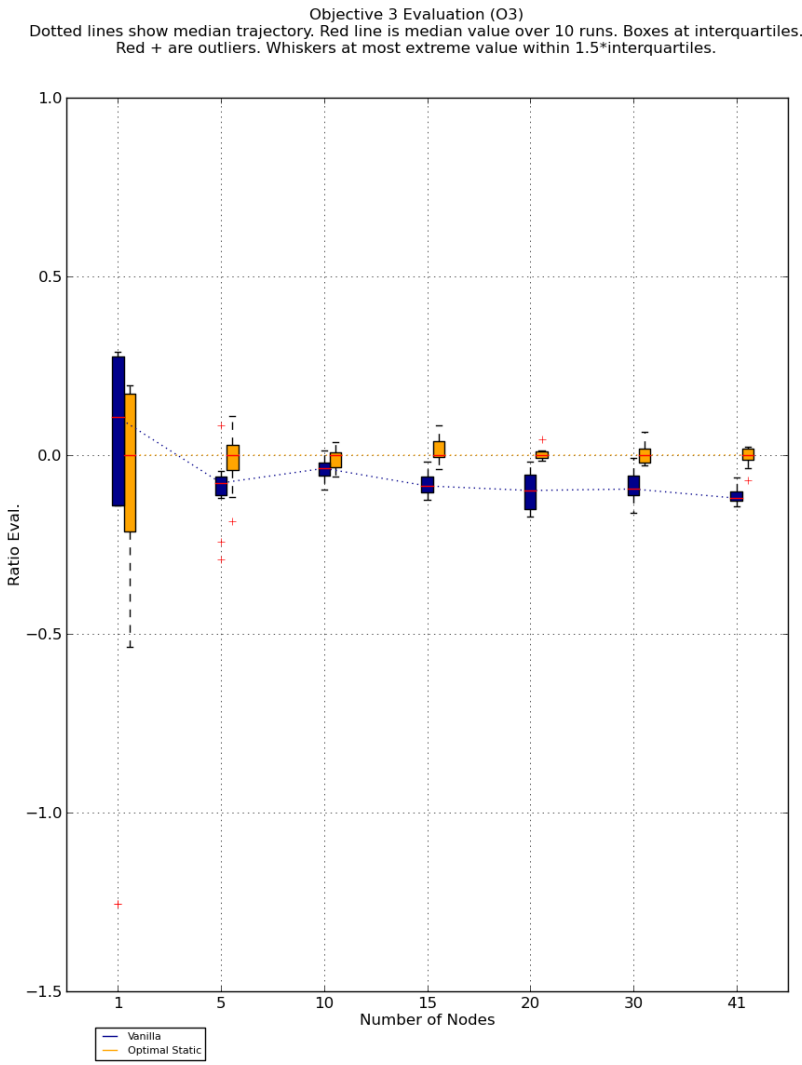


(b) Number of signatures distributed during experiment.

Figure E.5 – Metric observations.



(a) Immunisation rate (O1), combining metric M1,M2 data



(b) Immunisation rate suitability upon enterprise networks (O3), combining metric M1,M2,M3 data

Figure E.6 – Box plot observations of our Self-Healing System research objective evaluations (O1 and O3) as tested on an enterprise network. Using our extension of the Weisbin & Rodriguez ratio equation, with the SendAll algorithm results used as reference. Greater values imply better performance. Note that plot y-axes are truncated to -1.5, in order to exclude Optimal-Static's outlier value transposed from the M2 (Time) metric results.

E.4 Time Delay: Bash Background Process Execution Order

Here we show execution order of background processes and foreground processes is non-deterministic while using the bash `eval` function. We use this approach to initialise each instance of CARDINAL-Vanilla within our virtual network tests and use this output to justify the use of a time delay to ensure the execution order is predetermined.

```
#!/bin/bash
# File name: test_bash_queue_execution_order.sh
#
HOSTS=10
for(( h=1; h<=$HOSTS; h++ ))
do
        if [ $h -eq $HOSTS ]     # Final host:
        then
                eval "echo $h"   # eval as a foreground process
        else
                eval "echo $h &" # eval as a background process
        fi
        # location of possible time delay
done

#--------------------------
#       Output:
#--------------------------
# bash --version
GNU bash, version 4.2.45(1)-release (x86_64-pc-linux-gnu)
#
#
# ./test_bash_queue_execution_order.sh
1
2
3
4
5
6
7
8
10
9
#£
```

Appendix F

Parameter Range Evaluations

F.1 Combined System Performance Measures: Part 2

This section describes an approach to combining measures to enable a multi-objective system performance evaluation. It complements section 6.10.

Section F.1.1 describes a bit-based extension that is used by the search evaluation function in 8.4 and 8.6. This combination approach inaccurately combines metrics with a MIN-type preferred semantic condition. To remove the effect, each search candidate is re-evaluated using the accurate ratio of difference evaluation measure, specified in 6.10.3.

F.1.1 Failed Route: Log Ratio with Inverse

With respect to our goals for Objective 3 (O3) evaluation, two deficiencies of Rodriguez's equations were identified and described, alongside the introduction of their equations in subsection 6.10.2. We shall describe extensions to handle those additions. First, by adding an inverse to handle the MIN semantic measure condition case:

$$P^{''}(k,m) = \log_2 \left[{}^{1}/\frac{p(k,m)}{p(k,1)} \right] \tag{F.1}$$

such that both MIN and MAX cases can be handled by $P(k,m,c_k)$:

$$P(k,m,c_k) = \begin{cases} P^{'}(k,m), & \text{if } c_k = MAX \\ P^{''}(k,m), & \text{if } c_k = MIN \end{cases} \tag{F.2}$$

To include the measure-specific weightings (required for our third objective (O3)) a weight w_k is multiplied against the logarithm result. The updated performance measure functions become:

$$P^{'}(k,m,w_k) = \log_2 \left[\frac{p(k,m)}{p(k,1)} \right] * w_k \tag{F.3}$$

$$P^{''}(k,m,w_k) = \log_2 \left[{}^{1}/\frac{p(k,m)}{p(k,1)} \right] * w_k \tag{F.4}$$

Our new weighting parameter w_k is then added to the P functions in a straight-forward manner,

$$P(k,m,w_k,c_k) = \begin{cases} P^{'}(k,m,w_k), & \text{if } c_k = MAX \\ P^{''}(k,m,w_k), & \text{if } c_k = MIN \end{cases} \tag{F.5}$$

and therefore Equation 6.12, our scoring function becomes:

$$s(m) = \frac{1}{2}\sum_{k=1}^{N} P(k, m, w_k, c_k) \tag{F.6}$$

Summary

We have shown our own extension to Rodriguez and Weisbin's system performance evaluation (Rodriguez & Weisbin, 2003), using an inverse ratio, weights, conditional function and maintaining the Rodriguez's use of binary units as is summarised by Equation F.6. In the following section we will show an example of the pitfalls when evaluating system performances using this described extension to Rodriguez's approach.

Example Calculation of Objective 3 (O3)

We will repeat the simple example from section 6.10.3 showing how the weighted Log Ratios with Inverses equations perform with the example systems presented in Table 6.12. The example systems exemplify distance equal and scaling behaviour of the measure equations, as described in more detail above. In this case, we start by enter each system column of performance measure values (e.g. $[10, 10, 10]$) with their respective weights and condition values into Equation F.5.

Table F.1 shows the output of this first function for each system measurement. With $k = 1$ and $k = 2$ we expect counteracting values for a given system (e.g. $[-0.584, 0.584]$). This is due to the equal weighting, opposing semantics and equal values for the reference system (e.g. $[15, 15)$ and each test system (e.g. $[10, 10]$). We then find that the third measure output to be one-tenth that of the second measure value. The reference system evaluates to 0 for each of its measures, as it is using its own results to create the ratio giving 1, and $log_2 1$ then evaluates to 0.

Performance Measure (k)	Systems M_1	M_2	M_3	M_4	M_5	M_6	M_7	Weighting	Condition
1	0.0	-0.584	0.415	0.093	0.180	0.263	0.341	1	MAX
2	0.0	0.584	-0.415	0.447	0.321	0.206	0.099	1	MIN
3	0.0	0.058	-0.041	0.0	0.0	0.0	0.0	0.1	MIN

Table F.1 – Table of results from our $P(k, m, w_k, c_k)$ performance measure function in Equation F.5 for individual measure data from Table 6.12. Results are truncated at 3 decimal places.

Alarmbells 1/2

Importantly, we find that distance information between the systems is not maintained into the new ratio space. This is exemplified in our first three test systems by their equal distance (of 5) in result space, but differing ratio distances in ratio space. From this point onward, we can only draw interval distance comparisons on ratio score pairs within the same k (measure source) and that are both positive or both negative.

Next we can see the final results of Equation F.6 in Table F.2, which sums the $P(k, m, w_k, c_k)$ function outputs and multiplies by .5 per system.

Alarmbells 2/2

Table F.2 shows that multiplication by .5 has reduced the ratio space distance difference between systems M_1 and M_2. Systems $M_4, ..M_7$ each have equally improved distances and equal weighting, yet there is transparent discrepancy between those values, both in logarithm-ratio space (Table F.1) and in

Performance Measure	Systems M_1	M_2	M_3	M_4	M_5	M_6	M_7
O3	0.0	0.029248	-0.020751	0.270284	0.251250	0.234742	0.220286

Table F.2 – Table of results from $s(m)$ score function in Equation F.6. Results are truncated at 6 decimal places. Noticeable differences in O3 values for systems $M_4, ..M_7$ of equal weight and improved distance.

score space (as an outcome of Equation F.6). This difference is exacerbated (increased or reduced) as the configurations of weights, conditions, absolute value differences and signed value differences change.

Summary

We have shown that the latter four systems scale differently with differing values of the MAX and MIN condition measures. They have clear final score $s(m)$ variation despite having equally distanced measures. Using the Log Ratios with Inverses is not a suitable reverse alternative to a ratio, and particularly so when combining values with opposing MIN/MAX semantics as a system performance measure.

F.2 Descriptive Statistics of Each Parameter Value Range

Table F.6 and Table F.7 show the complete set of summary statistics of metrics for each parameter value change.

F.3 Box Plots of Each Parameter Value Range

The following figures show the box plot performance per parameter per range value, see Figure F.1, Figure F.2, Figure F.3, Figure F.4, Figure F.5. The list of metrics are identified in Table 8.2 and complement the descriptive statistic result tables per metric, which are also within this appendix.

F.4 Ordinal Correlation from Parameter Value to Metric Results

Tables F.3, F.4, F.5, F.9 and F.10 show the *rho* statistic and *P-value* results from the Spearman's rank correlations described in subsection 8.3.1.

v.	*rho*	$P - value$	*df*	Correlation
Under Attack Volume Percentage: P0				
Inputs	nan	nan	6	No effect
Detectors	nan	nan	6	No effect
Distributed-M1	0.991	1.46^{-05}	6	Sig. $<$ 0.05 Pos.
Response-Total	1	0	6	Sig. $<$ 0.05 Pos.
Response-μ	0.937	0.00185	6	Sig. $<$ 0.05 Pos.
Time-M2	-0.964	0.000454	6	Sig. $<$ 0.05 Neg.
IO-Events-Total	0.929	0.00252	6	Sig. $<$ 0.05 Pos.
IO-Events-μ	0.982	8.29^{-05}	6	Sig. $<$ 0.05 Pos.
DataSent-M3	1	0	6	Sig. $<$ 0.05 Pos.
DataSent-μ	1	0	6	Sig. $<$ 0.05 Pos.

Table F.3 – Table of Spearman's rank *rho* (monotonic) correlations between the P0 range values and the median results for each metric listed.

v.	*rho*	$P - value$	*df*	Correlation
Initial Priority Value Multiplier: P1				
Inputs	nan	nan	7	No effect
Detectors	nan	nan	7	No effect
Distributed-M1	0.946	0.000386	7	Sig. $<$ 0.05 Pos.
Response-Total	0.833	0.0102	7	Sig. $<$ 0.05 Pos.
Response-μ	0.976	3.31^{-05}	7	Sig. $<$ 0.05 Pos.
Time-M2	-0.929	0.000863	7	Sig. $<$ 0.05 Neg.
IO-Events-Total	-0.0479	0.91	7	Insig.
IO-Events-μ	nan	nan	7	No effect
DataSent-M3	1	0	7	Sig. $<$ 0.05 Pos.
DataSent-μ	1	0	7	Sig. $<$ 0.05 Pos.

Table F.4 – Table of Spearman's rank *rho* (monotonic) correlations between the P1 range values and the median results for each metric listed.

	Qty of Modules						Responses				Time to Send		Qty of I/O Events				MiB Sent			
	Inputs		Detectors		Dist. M1		Median Host		Total		M2		Total		Median Host		Total M3		Median Host	
v	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)
Under Attack Volume Percentage: P0																				
0.25	13	0	13	0	2	0.894	14798.5	19.2	72747	138	58.568	2.69	120	0	24	0	1.7	0.064	0.4	0
0.33	13	0	13	0	7	1.02	14863	14.7	73609	169	40.332	13.7	213	2.62	43	0.7	4.05	0.134	0.9	0.064
0.5	13	0	13	0	8.5	1.51	14889	19.6	74117	201	34.3185	9.63	211	1.85	43	0.539	5.6	0.211	1.2	0.06
0.66	13	0	13	0	11.5	1.02	14926.5	26.3	74438.5	119	32.595	2.11	309	1.2	62	1.12	10.3	0.545	2.1	0.0922
0.75	13	0	13	0	12.5	1.3	14953.5	12.3	74528.5	129	29.303	4.15	310	2.68	63	0.872	11.85	0.699	2.4	0.118
0.9	13	0	13	0	12.5	0.781	14941	20.5	74601.5	126	31.639	3.84	407	3	83	0.4	16.6	0.655	3.4	0.11
1	13	0	13	0	13	0.4	14953.5	13.2	74696.5	45.9	28.5995	2.79	405	4.4	83	1.2	18.6	0.415	3.8	0.064
Initial Priority Value Multiplier: P1																				
0.5	13	0	13	0	10	1.36	14895	24	74409	206	33.6515	3.99	309	2.87	63	0.6	9.35	0.326	1.9	0.08
1	13	0	13	0	10	1.11	14921.5	16.6	74450	151	33.768	2.66	310.5	5.02	63	0.7	10.25	0.492	2.1	0.0748
1.5	13	0	13	0	11.5	1	14922.5	19.6	74452.5	139	33.307	2.18	312	3.94	63	1.62	10.8	0.498	2.25	0.0781
2	13	0	13	0	12	0.781	14927	16.4	74518.5	102	31.6795	4.13	309.5	3.67	63	0.64	11.4	0.503	2.35	0.102
2.5	13	0	13	0	12	0.64	14936.5	19.5	74419.5	214	32.489	4.32	308.5	5.53	63	0.539	11.55	0.753	2.4	0.118
3	13	0	13	0	13	0.64	14935.5	22.1	74535.5	140	28.128	3.88	310.5	4.66	63	0.806	12.55	0.533	2.5	0.136
3.5	13	0	13	0	12.5	0.781	14953	15.2	74522.5	121	30.097	4.58	307	3.32	63	0.458	12.95	0.38	2.7	0.102
4	13	0	13	0	13	0.49	14959.5	12.5	74652.5	102	25.546	4.37	311	1.3	63	0.632	14.25	0.465	2.85	0.0917
Priority Value Suppression: P2																				
0.1	13	0	13	0	13	0	14967.5	13.6	74570.5	131	27.275	3.79	310	1.97	63	0.3	22.25	1.33	4.6	0.293
0.2	13	0	13	0	13	0	14974.5	8.34	74539	153	25.961	2.78	310.5	1.94	63	0.447	23.75	1.14	4.8	0.232
0.3	13	0	13	0	13	0	14969.5	13.5	74626.5	111	27.7855	3.64	309.5	3.4	63	0.8	23.15	0.815	4.7	0.169
0.4	13	0	13	0	13	0	14960	9.22	74483	102	30.3195	3.69	307	2.82	63	0.539	21.65	1.08	4.45	0.228
0.5	13	0	13	0	13	0	14971	15.7	74589	128	27.156	2.97	309.5	3.43	63	1	21.35	1.04	4.3	0.202
0.6	13	0	13	0	13	0	14958.5	16.4	74695	105	29.053	2.77	308	1.87	63	0.922	18.65	0.908	3.8	0.241
0.7	13	0	13	0	13	0.3	14953	13.2	74365.5	522	31.1105	4.03	308	4.44	63	0.632	17.3	1.26	3.35	0.224
0.8	13	0	13	0	13	0	14959	18.9	74639	95.4	29.4215	2.51	308.5	4.36	63	0.781	14.95	0.953	3.05	0.195
0.9	13	0	13	0	12.5	0.872	14941	22.4	74616.5	134	28.4775	3.19	311	2.54	63	0.3	12.65	0.842	2.65	0.174
1	13	0	13	0	11	1.49	14934	14.4	74485.5	110	32.1575	2.74	311.5	2.86	63	0.917	9.45	0.31	2	0.0748
1.25	13	0	13	0	3.5	1.28	14842	45.6	74015.5	3290	62.9235	1.5	310	3.01	63	0.8	4.4	0.294	0.9	0.064
1.5	13	0	13	0	2.5	0.917	14850.5	17.3	73870	123	60.5815	1.65	308.5	3.47	63	0.671	3.6	0.214	0.8	0.049
1.75	13	0	13	0	1	0.748	14807	29.9	73621.5	160	60.2395	2.18	306.5	4.07	62.5	0.98	3.15	0.174	0.7	0.049
2	13	0	13	0	1.5	1.11	14842.5	17.4	73640	127	61.4	2.35	309	3.87	63	0.632	3.1	0.133	0.7	0.0458

Table F.6 – Table shows measured results upon the variation of only the value v setting for parameters P0, P1 and P2. Tests are conducted under a given virtual network experiment set-up with five virtual hosts. Median μ and sampled standard deviation (*Std.*) summary statistics are reported per metric.

	Qty of Modules						Responses				Time to Send		Qty of I/O Events				MiB Sent			
	Inputs		Detectors		Dist. M1		Median Host		Total		M2		Total		Median Host		Total M3		Median Host	
v	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)	μ	(*Std.*)
Static Moving Window Size: P3																				
1	13	0	13	0	7.5	1.36	14911	15.6	74385	205	29.451	16.2	185.5	7.2	37	1.57	3.05	0.335	0.7	0.07
2	13	0	13	0	7.5	1.43	14910	11.7	74342	89.3	34.5395	12.5	188	5.1	38.5	1.97	3.2	0.245	0.65	0.0917
3	13	0	13	0	8	1.36	14905	12.2	74386	75.5	31.8435	3.7	185	7.31	36	1.92	3.3	0.26	0.7	0.104
4	13	0	13	0	8.5	1.66	14914.5	17.2	74493.5	106	29.512	14.7	191	5.7	38.5	1.57	3.7	0.293	0.8	0.07
5	13	0	13	0	7.5	1.96	14915.5	10.6	74439.5	151	32.216	14.6	190.5	6.96	38.5	2.05	3.55	0.316	0.8	0.098
6	13	0	13	0	8.5	1.51	14915.5	16.3	74432.5	99.2	32.5825	14	185	10.7	37.5	2.23	3.5	0.345	0.8	0.0748
8	13	0	13	0	8.5	1.72	14907	19.5	74358	180	31.759	14.3	193	10.3	38.5	2.21	3.9	0.405	0.9	0.12
10	13	0	13	0	9.5	1.22	14921	15.7	74449	72.1	29.5555	4.93	205.5	7.7	40	2.06	4.6	0.309	1	0.0943
12	13	0	13	0	12	0.9	14944	11.5	74524.5	60	29.576	2.76	310.5	3.69	63	0.917	9.95	0.504	2.1	0.0943
14	13	0	13	0	12	0.872	14939	16.9	74532	416	31.1565	4.74	310.5	3.03	63	0.748	10.3	0.402	2.2	0.08
16	13	0	13	0	12	0.748	14956.5	22.3	74591	105	28.222	2.56	309.5	3.57	63	0.917	11	0.666	2.25	0.127
18	13	0	13	0	12	0.943	14937.5	20.8	74555	124	29.3675	3.93	310.5	3.35	63	0.748	10.65	0.514	2.2	0.108
20	13	0	13	0	12	0.64	14945.5	15.3	74564	134	28.752	3.6	310.5	4.15	63	0.8	10.85	0.547	2.25	0.11
40	13	0	13	0	12	0.775	14944	15.9	74431	122	30.4865	3.62	308.5	2.84	63	1.1	11.25	0.299	2.3	0.0748
60	13	0	13	0	12	0.781	14930.5	19.5	74462	126	32.6815	3.08	307.5	3.51	63	0.781	11.45	0.437	2.3	0.0872
80	13	0	13	0	12	0.748	14949	14.3	74557.5	90.9	28.235	2.45	309	2.68	63	0.8	11.7	0.517	2.4	0.14
100	13	0	13	0	12.5	1.14	14945	14.6	74525	194	29.511	5.23	309	3.29	63	0.9	11.45	0.739	2.3	0.194
Dendritic Cell Lifespan: P4																				
1	13	0	13	0	12	0.8	14933.5	38.1	74462	147	28.76	3.5	310.5	2.46	63	0.539	11.4	0.316	2.35	0.0663
5	13	0	13	0	12	0.943	14926	31.5	74441.5	205	30.4255	2.92	311	2.73	63	0.458	11.15	0.633	2.3	0.16
10	13	0	13	0	11	1.2	14945	27.7	74586	141	29.649	1.59	308.5	3.83	63	0.917	11.35	0.308	2.4	0.0539
20	13	0	13	0	12.5	0.663	14944	18.8	74572.5	157	27.5645	4.16	309	3.31	62	0.6	12	0.42	2.4	0.08
40	13	0	13	0	13	0.458	14952.5	22.6	74442.5	82.2	29.7925	4.06	307.5	2.82	63	1.14	11.25	0.525	2.3	0.09
50	13	0	13	0	13	0.917	14953	22.2	74556.5	158	30.065	3.25	311	2.38	63	0.781	11.45	0.498	2.35	0.0872
60	13	0	13	0	12	0.7	14945	10.9	74625	167	29.706	3.04	307.5	4.03	62	0.943	11.55	0.461	2.4	0.1
80	13	0	13	0	12	1	14946	18.4	74517	210	30.688	3.19	310.5	3.06	63	0.64	11.35	0.438	2.4	0.11
100	13	0	13	0	13	0.663	14945	18.7	74504.5	156	28.849	2.47	308.5	2.77	62	1	11.6	0.544	2.4	0.125

Table F.7 – Table shows measured results upon the variation of only the value v setting for parameters P3 and P4. Tests are conducted under a given virtual network experiment set-up with five virtual hosts. Median μ and sampled standard deviation (*Std.*) summary statistics are reported per metric.

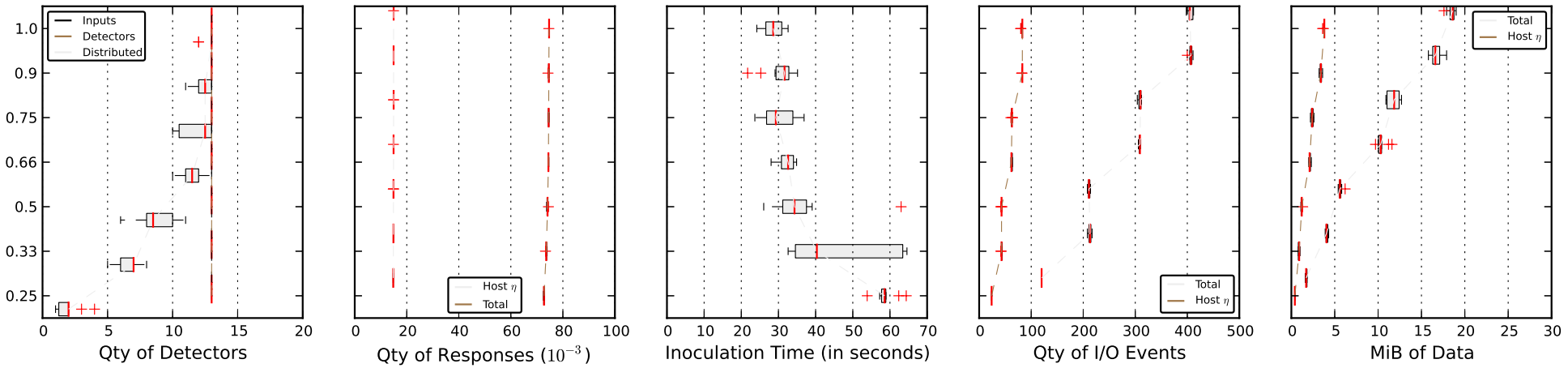


Figure F.1 – Under Attack Volume Percentage (P0) - parameter value range effects.

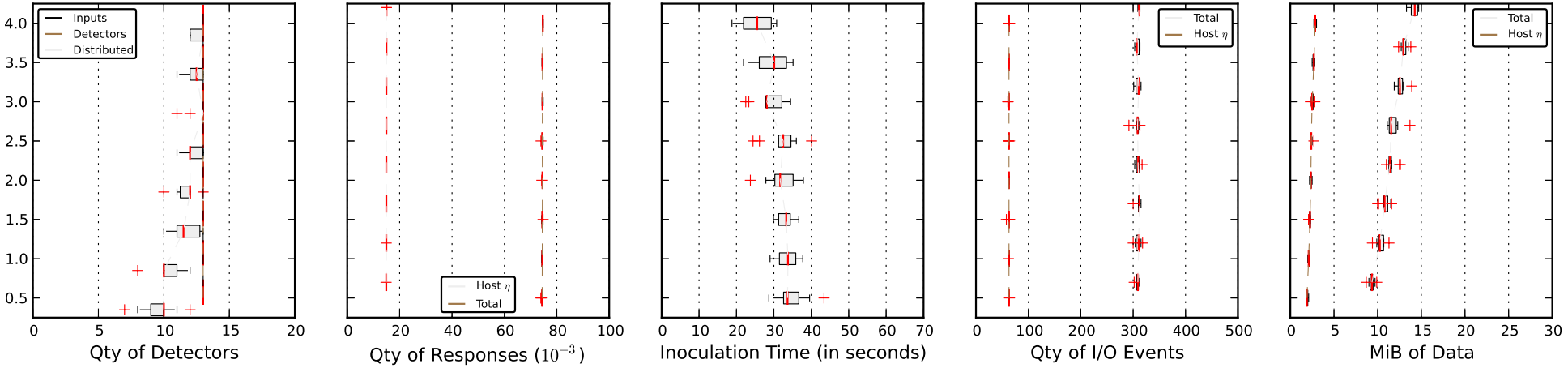


Figure F.2 – Initial Priority Value Multiplier (P1) - parameter value range effects.

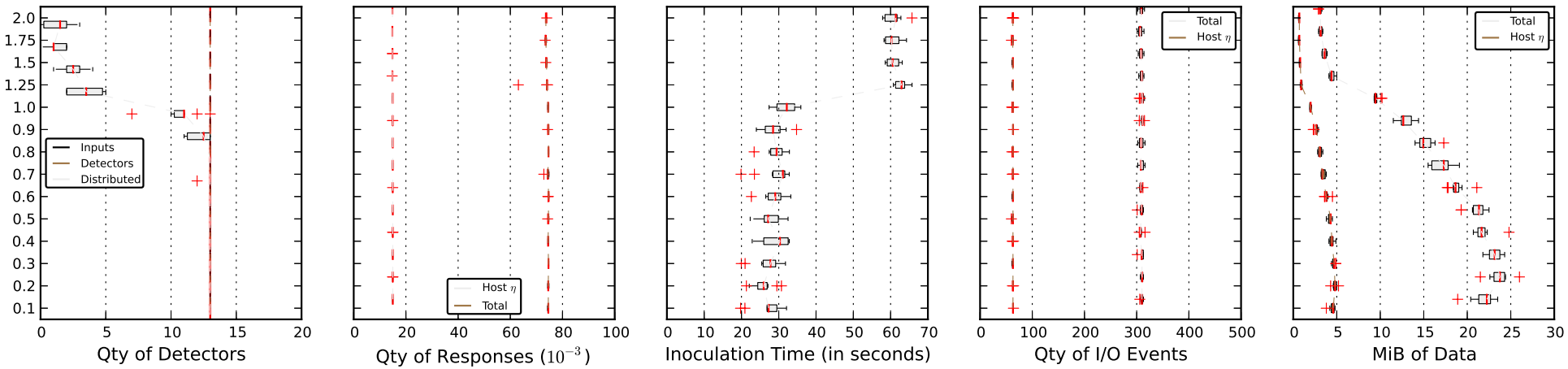


Figure F.3 – Priority Value Suppression (P2) - parameter value range effects.

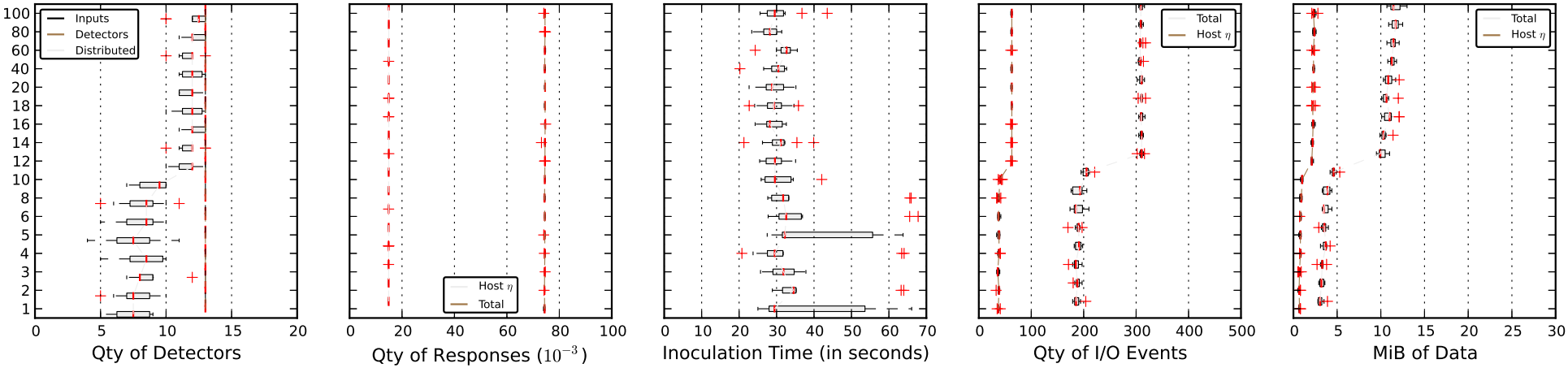


Figure F.4 – Static Moving Window Size (P3) - parameter value range effects.

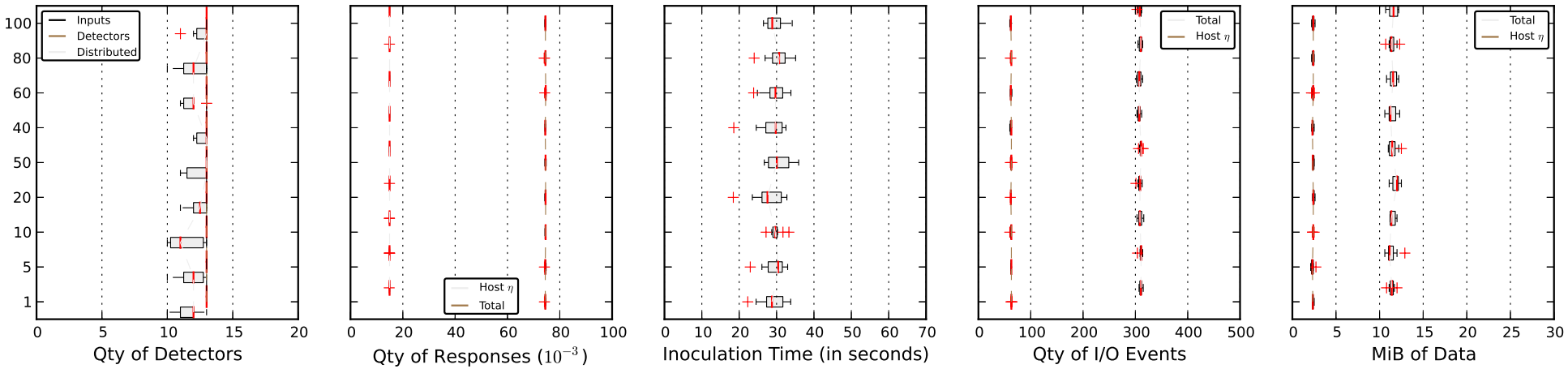


Figure F.5 – Dendritic Cell Lifespan (P4) - parameter value range effects.

v.	*rho*	$P-value$	*df*	Correlation
Priority Value Suppression: P2 (Full Range)				
Inputs	nan	nan	13	No effect
Detectors	nan	nan	13	No effect
Distributed-M1	-0.898	1.29^{-05}	13	Sig. $<$ 0.05 Neg.
Response-Total	-0.613	0.0197	13	Sig. $<$ 0.05 Neg.
Response-μ	-0.921	2.98^{-06}	13	Sig. $<$ 0.05 Neg.
Time-M2	0.855	9.77^{-05}	13	Sig. $<$ 0.05 Pos.
IO-Events-Total	-0.168	0.566	13	Insig.
IO-Events-μ	-0.378	0.182	13	Insig.
DataSent-M3	-0.987	7.38^{-11}	13	Sig. $<$ 0.05 Neg.
DataSent-μ	-0.986	1.2^{-10}	13	Sig. $<$ 0.05 Neg.

Table F.5 – Table of Spearman's rank *rho* (monotonic) correlations between the full P2 range values and the median results for each metric listed. The full range tested here is between 0.1 and 2.0.

v.	*rho*	$P-value$	*df*	Correlation
Priority Value Suppression: P2 (reduced range)				
Inputs	nan	nan	9	No effect
Detectors	nan	nan	9	No effect
Distributed-M1	-0.701	0.024	9	Sig. $<$ 0.05 Neg.
Response-Total	0.0545	0.881	9	Insig.
Response-μ	-0.842	0.00222	9	Sig. $<$ 0.05 Neg.
Time-M2	0.685	0.0289	9	Sig. $<$ 0.05 Pos.
IO-Events-Total	0.171	0.637	9	Insig.
IO-Events-μ	nan	nan	9	No effect
DataSent-M3	-0.964	7.32^{-06}	9	Sig. $<$ 0.05 Neg.
DataSent-μ	-0.964	7.32^{-06}	9	Sig. $<$ 0.05 Neg.

Table F.8 – Table of Spearman's rank *rho* (monotonic) correlations between the reduced P2 range of values and the median results for each metric listed. The tested range here is between 0.1 and 1.0, a more intuitive range.

v.	*rho*	$P-value$	*df*	Correlation
Static Moving Window Size: P3				
Inputs	nan	nan	16	No effect
Detectors	nan	nan	16	No effect
Distributed-M1	0.926	9.46^{-08}	16	Sig. $<$ 0.05 Pos.
Response-Total	0.662	0.00381	16	Sig. $<$ 0.05 Pos.
Response-μ	0.82	5.65^{-05}	16	Sig. $<$ 0.05 Pos.
Time-M2	-0.392	0.119	16	Insig.
IO-Events-Total	0.714	0.0013	16	Sig. $<$ 0.05 Pos.
IO-Events-μ	0.882	2.81^{-06}	16	Sig. $<$ 0.05 Pos.
DataSent-M3	0.979	1.06^{-11}	16	Sig. $<$ 0.05 Pos.
DataSent-μ	0.98	7.09^{-12}	16	Sig. $<$ 0.05 Pos.

Table F.9 – Table of Spearman's rank *rho* (monotonic) correlations between the P3 range values and the median results for each metric listed.

v.	*rho*	$P - value$	*df*	Correlation
Dendritic Cell Lifespan: P4				
Inputs	nan	nan	8	No effect
Detectors	nan	nan	8	No effect
Distributed-M1	0.443	0.232	8	Insig.
Response-Total	0.217	0.576	8	Insig.
Response-μ	0.593	0.0922	8	Insig.
Time-M2	0.233	0.546	8	Insig.
IO-Events-Total	-0.432	0.245	8	Insig.
IO-Events-μ	-0.456	0.217	8	Insig.
DataSent-M3	0.326	0.391	8	Insig.
DataSent-μ	0.461	0.212	8	Insig.

Table F.10 – Table of Spearman's rank *rho* (monotonic) correlations between the P4 range values and the median results for each metric listed.

F.5 Descriptive Statistics of Immunisation Rates

Table F.11 shows summary statistics of each system performance measure for each parameter value change.

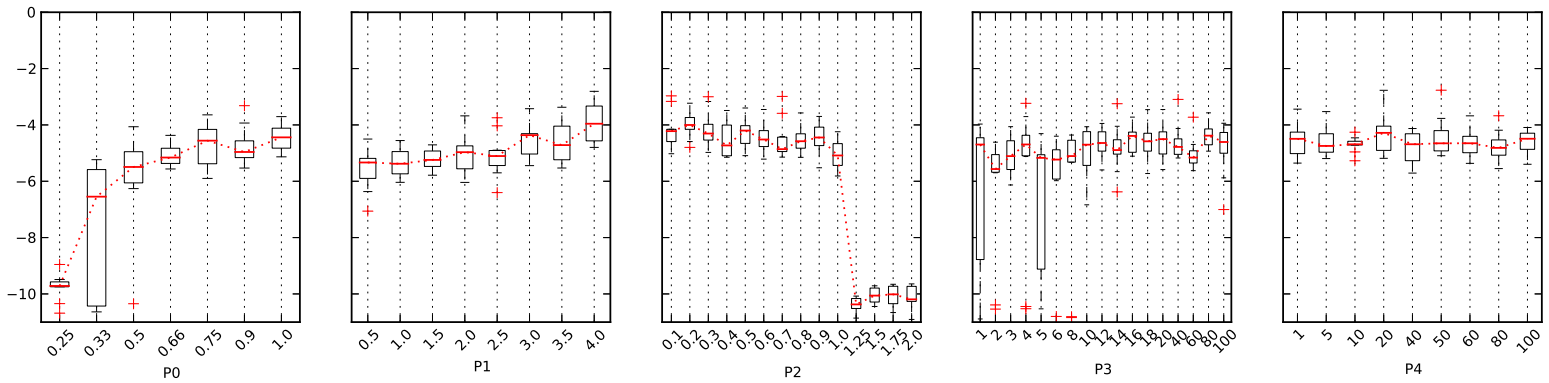
F.6 Box Plots of Immunisation Rates

Figure F.6 shows box-plot results of three system performance measures for each parameter range.

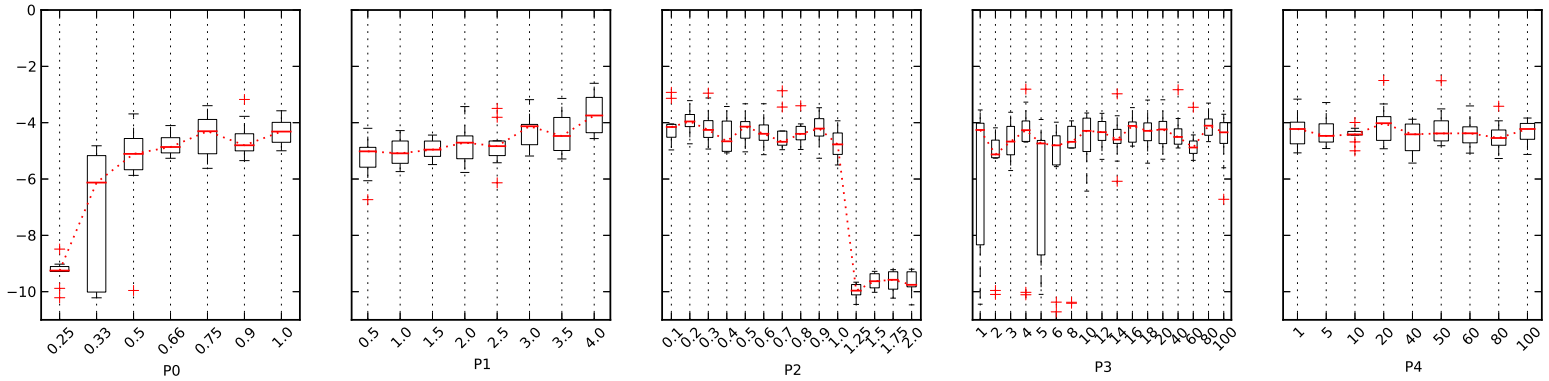
F.7 Best of Search States from Virtual Network Environment Tests

Metric and immunisation rate scores of the best configurations discovered during the virtual network parameter tuning search are collected in the following tables:

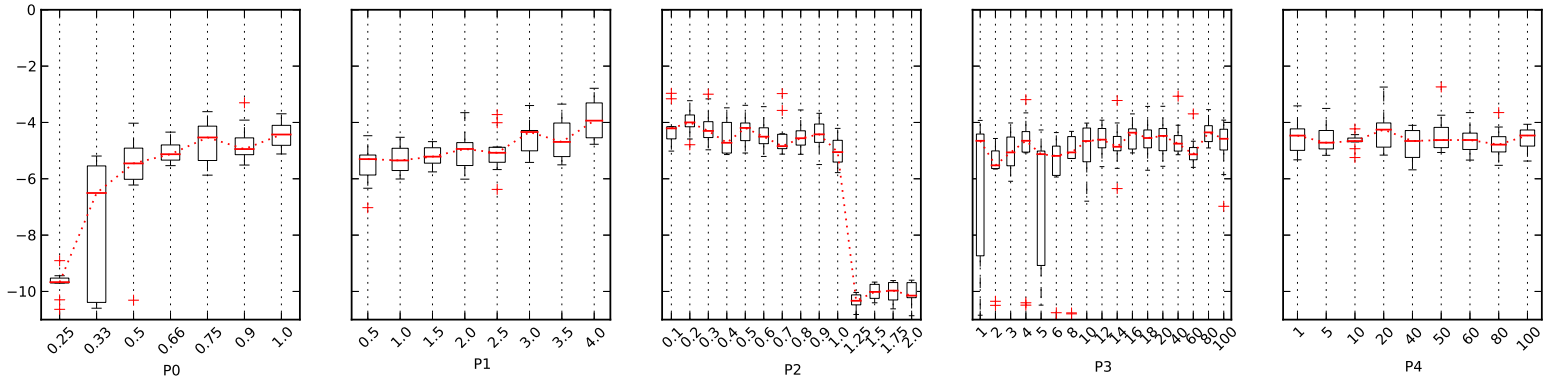
- Tables F.12 and F.13 are ranked by the generalised immunisation rate (O1).
- Results within the corresponding experiment chapter in tables 8.11 and 8.12 are ranked by the immunisation rate for low through-put networks (O2).
- Finally, in tables F.14 and F.15 results are ranked by the immunisation rate for high through-put networks (O3).



(a) For transmission-agnostic evaluation (O1).



(b) For evaluation on low-throughput networks (O2).



(c) For evaluation on high-throughput networks (O3).

Figure F.6 – The box-plots show results of our three system performance measures (scored on the *Y-axis*) from trials with our five parameter's sets of configuration values (*X-axis*). Greater *Y-axis* values are preferred. The red dotted line shows the median trend. From left to right, P0, P1, P2, P3 and P4 parameter results are shown. Each are described within the chapter.

v	O1 μ	O1 (*Std.*)	O2 μ	O2 (*Std.*)	O3 μ	O3 (*Std.*)
Under Attack Volume Percentage: P0						
0.25	-9.72	0.444	-9.25	0.445	-9.67	0.444
0.33	-6.55	2.3	-6.13	2.3	-6.5	2.3
0.5	-5.49	1.63	-5.11	1.63	-5.45	1.63
0.66	-5.16	**0.362**	-4.86	**0.353**	-5.13	**0.361**
0.75	-4.56	0.702	**-4.3**	0.694	-4.53	0.701
0.9	-4.96	0.635	-4.8	0.627	-4.94	0.634
1.0	**-4.44**	0.467	-4.32	0.464	**-4.43**	0.467
Initial Priority Value Multiplier: P1						
0.5	-5.33	0.691	-5.02	0.687	-5.3	0.691
1.0	-5.38	0.458	-5.09	0.452	-5.35	0.457
1.5	-5.24	**0.356**	-4.96	**0.349**	-5.21	**0.356**
2.0	-4.97	0.694	-4.71	0.689	-4.94	0.694
2.5	-5.1	0.732	-4.83	0.722	-5.08	0.731
3.0	-4.36	0.643	-4.13	0.638	-4.34	0.643
3.5	-4.72	0.76	-4.47	0.756	-4.69	0.759
4.0	**-3.96**	0.729	**-3.74**	0.729	**-3.94**	0.729
Priority Value Suppression: P2						
0.1	-4.22	0.631	-4.15	0.623	-4.21	0.63
0.2	**-4**	0.463	**-3.96**	0.451	**-4**	0.462
0.3	-4.31	0.607	-4.26	0.614	-4.3	0.608
0.4	-4.73	0.615	-4.66	0.602	-4.72	0.614
0.5	-4.2	0.495	-4.14	0.489	-4.19	0.494
0.6	-4.52	0.462	-4.39	0.469	-4.5	0.463
0.7	-4.86	0.67	-4.68	0.656	-4.84	0.669
0.8	-4.58	0.419	-4.4	0.411	-4.56	0.418
0.9	-4.45	0.543	-4.21	0.534	-4.42	0.542
1.0	-5.08	0.473	-4.77	0.469	-5.05	0.472
P2-Full						
1.25	-10.4	**0.239**	-9.96	**0.241**	-10.3	0.24
1.5	-10.1	0.259	-9.63	0.262	-10	**0.259**
1.75	-10	0.366	-9.58	0.367	-9.97	0.366
2.0	-10.2	0.382	-9.76	0.383	-10.1	0.382

(a)

v	O1 μ	O1 (*Std.*)	O2 μ	O2 (*Std.*)	O3 μ	O3 (*Std.*)
Static Moving Window Size: P3						
1	-4.7	2.73	-4.26	2.73	-4.65	2.73
2	-5.57	2.1	-5.13	2.1	-5.52	2.1
3	-5.11	0.642	-4.67	0.64	-5.06	0.642
4	-4.69	2.49	-4.27	2.5	-4.65	2.49
5	-5.17	2.48	-4.74	2.48	-5.13	2.48
6	-5.23	2.37	-4.8	2.37	-5.19	2.37
8	-5.11	2.41	-4.68	2.41	-5.06	2.41
10	-4.7	0.85	-4.29	0.848	-4.66	0.85
12	-4.64	0.467	-4.34	0.461	-4.61	0.466
14	-4.89	0.803	-4.6	0.797	-4.86	0.802
16	-4.39	0.437	-4.12	0.434	-4.36	0.437
18	-4.58	0.671	-4.29	0.665	-4.55	0.67
20	-4.5	0.609	-4.24	0.603	-4.48	0.608
40	-4.78	0.595	-4.51	0.592	-4.75	0.595
60	-5.16	0.522	-4.88	0.52	-5.13	0.522
80	**-4.38**	**0.41**	**-4.11**	**0.411**	**-4.35**	**0.41**
100	-4.61	0.898	-4.34	0.889	-4.58	0.897
Dendritic Cell Lifespan: P4						
1	-4.49	0.586	-4.22	0.584	-4.47	0.586
5	-4.75	0.492	-4.47	0.482	-4.72	0.491
10	-4.69	**0.261**	-4.42	**0.257**	-4.66	**0.26**
20	**-4.28**	0.695	**-4.02**	0.696	**-4.26**	0.695
40	-4.69	0.558	-4.41	0.55	-4.66	0.557
50	-4.65	0.679	-4.38	0.671	-4.63	0.678
60	-4.65	0.512	-4.38	0.511	-4.62	0.512
80	-4.81	0.536	-4.54	0.529	-4.79	0.535
100	-4.49	0.416	-4.22	0.41	-4.46	0.415

(b)

Table F.11 – Tables of system performance measure results taken from 10 repeated trials with changes to each parameter's configuration value (v.). Median (μ) and standard deviations (*Std.*) are shown for each of the three performance measure objectives O1, O2 and O3. Greatest median and smallest *Std.* values are preferred and have been emboldened. *P2-Full* has been segregated as this range of values show particularly poor performance, as discussed in Part 1.

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	0.66	2	0.1	2	19.500	0.500	1.000	**14.889**	2.246	2.835	46.650	3.939	7.150	2
2	1.0	3.5	0.1	1	19.500	1.732	1.000	14.974	2.563	4.075	61.500	15.147	10.400	7
3	1.0	1.0	0.1	1	**20.000**	3.528	1.000	15.271	71.078	2.943	60.550	14.126	17.850	8
4	1.0	1.0	0.1	1	**20.000**	0.663	0.750	15.473	7.904	9.268	70.400	14.602	16.750	9
5	0.75	2.0	0.1	1	**20.000**	0.458	0.750	15.712	1.368	2.486	46.000	4.619	4.425	4
6	0.75	2.5	0.1	2	**20.000**	0.458	0.750	15.806	1.991	2.217	55.350	12.394	5.075	2
7	0.75	3.5	0.2	1	19.000	1.432	0.750	16.017	9.418	9.769	50.950	15.067	30.025	6
8	0.75	2.0	0.1	1	19.000	0.490	1.000	16.103	4.840	2.894	42.650	13.007	14.150	4
9	0.75	2	0.95	1	19.000	1.414	1.000	16.109	4.779	2.835	**42.550**	13.512	15.100	0
10	0.75	0.5	0.1	1	19.000	0.458	0.750	16.133	3.236	3.056	49.050	4.820	4.775	6

Table F.12 – Table showing metric scores of the best configurations ranked by immunisation rate O1 score. Scores from the parameter tuning search under the virtual network environment set-up.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	0.66	2	0.1	2	**-1.99392**	0.37643	0.46621	**-2.48217**	0.40510	0.49479	**-2.03912**	0.37866	0.47236
2	1.0	3.5	0.1	1	-2.01883	0.45722	0.67821	-2.79025	0.47530	0.76950	-2.08645	0.44997	0.68672
3	1.0	1.0	0.1	1	-2.04525	11.93326	0.50921	-2.75625	12.09073	0.72437	-2.11635	11.94880	0.51251
4	1.0	1.0	0.1	1	-2.09142	1.32638	1.53962	-3.00325	1.52301	1.87300	-2.16002	1.34448	1.55388
5	0.75	2.0	0.1	1	-2.11858	0.22656	0.42683	-2.64167	0.23447	0.34158	-2.17048	0.22566	0.41126
6	0.75	2.5	0.1	2	-2.14675	0.32735	0.35879	-2.73550	0.47378	0.55579	-2.20485	0.33668	0.39009
7	0.75	3.5	0.2	1	-2.19442	1.60036	1.67825	-2.79717	1.39991	1.43696	-2.24632	1.57888	1.65868
8	0.75	2.0	0.1	1	-2.19633	0.80939	0.47612	-2.70067	0.87729	0.51117	-2.25233	0.81270	0.46875
9	0.75	2	0.95	1	-2.19733	0.82436	0.44750	-2.60208	0.86355	0.81237	-2.23218	0.82438	0.48117
10	0.75	0.5	0.1	1	-2.20125	0.53848	0.52812	-2.73625	0.57344	0.54021	-2.25475	0.54130	0.52122

Table F.13 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O1 score. Scores from the parameter tuning search under the virtual network environment set-up.

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	0.66	2	0.1	2	19.500	0.500	1.000	**14.889**	2.246	2.835	46.650	3.939	7.150	2
2	1.0	3.5	0.1	1	19.500	1.732	1.000	14.974	2.563	4.075	61.500	15.147	10.400	7
3	1.0	1.0	0.1	1	**20.000**	3.528	1.000	15.271	71.078	2.943	60.550	14.126	17.850	8
4	1.0	1.0	0.1	1	**20.000**	0.663	0.750	15.473	7.904	9.268	70.400	14.602	16.750	9
5	0.75	2.0	0.1	1	**20.000**	0.458	0.750	15.712	1.368	2.486	46.000	4.619	4.425	4
6	0.75	2.5	0.1	2	**20.000**	0.458	0.750	15.806	1.991	2.217	55.350	12.394	5.075	2
7	0.75	2	0.95	1	19.000	1.414	1.000	16.109	4.779	2.835	42.550	13.512	15.100	0
8	0.75	2.0	0.1	1	19.000	0.640	0.000	16.078	2.729	0.734	**38.750**	6.573	8.775	5
9	0.75	3.5	0.2	1	19.000	1.432	0.750	16.017	9.418	9.769	50.950	15.067	30.025	6
10	0.75	2.0	0.1	1	19.000	0.490	1.000	16.103	4.840	2.894	42.650	13.007	14.150	4

Table F.14 – Table showing metric scores of the best configurations ranked by immunisation rate O3 score. Scores from the parameter tuning search under the virtual network environment set-up.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	0.66	2	0.1	2	**-1.99392**	0.37643	0.46621	**-2.48217**	0.40510	0.49479	**-2.03912**	0.37866	0.47236
2	1.0	3.5	0.1	1	-2.01883	0.45722	0.67821	-2.79025	0.47530	0.76950	-2.08645	0.44997	0.68672
3	1.0	1.0	0.1	1	-2.04525	11.93326	0.50921	-2.75625	12.09073	0.72437	-2.11635	11.94880	0.51251
4	1.0	1.0	0.1	1	-2.09142	1.32638	1.53962	-3.00325	1.52301	1.87300	-2.16002	1.34448	1.55388
5	0.75	2.0	0.1	1	-2.11858	0.22656	0.42683	-2.64167	0.23447	0.34158	-2.17048	0.22566	0.41126
6	0.75	2.5	0.1	2	-2.14675	0.32735	0.35879	-2.73550	0.47378	0.55579	-2.20485	0.33668	0.39009
7	0.75	2	0.95	1	-2.19733	0.82436	0.44750	-2.60208	0.86355	0.81237	-2.23218	0.82438	0.48117
8	0.75	2.0	0.1	1	-2.21717	0.46889	0.12229	-2.51475	0.53059	0.32529	-2.23610	0.47378	0.11767
9	0.75	3.5	0.2	1	-2.19442	1.60036	1.67825	-2.79717	1.39991	1.43696	-2.24632	1.57888	1.65868
10	0.75	2.0	0.1	1	-2.19633	0.80939	0.47612	-2.70067	0.87729	0.51117	-2.25233	0.81270	0.46875

Table F.15 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O3 score. Scores from the parameter tuning search under the virtual network environment set-up.

Generation 1: Top 10 (A only)					
P0	0.5≥	**80%**	< 0.625 ≥	20%	<0.75
P1	0.5≥	50%	< 1.5 ≥	50%	<2.5
P2	0.7≥	30%	< 0.8 ≥	**70%**	<0.9
P3	1.0≥	50%	< 2.5 ≥	50%	<4.0

Generation 1: Top 10 (B only)					
P0	0.5≥	**60%**	< 0.75 ≥	40%	<1.0
P1	0.5≥	**80%**	< 2.0 ≥	20%	<3.5
P2	0.7≥	10%	< 0.8 ≥	**90%**	<0.9
P3	1.0≥	**70%**	< 2.5 ≥	30%	<4.0

All Generations Top 10s (A,B,B_2,B_3)					
P0	0.5≥	**60%**	< 0.75 ≥	40%	<1.0
P1	0.5≥	**62%**	< 2.25 ≥	38%	<4.0
P2	0.2≥	10%	< 0.55 ≥	**90%**	<0.9
P3	1.0≥	45%	< 3.0 ≥	**55%**	<5.0

Table F.16 – The table maps each of the elite configurations' parameter values into either the top or bottom regions of the parameter's range. The percentages show where those configuration values fell from each of the elite configurations. Shown are result sets A , B and a combination of A, B, B_2 and B_3 result sets. This table is included to give further informative coverage of the differing A and B set behaviours and an overview of all of the tested configuration sets.

F.8 Enterprise Network Environment Parameter Tuning Tests

This test ran for 6 days and consisted of 4 result sets, over 3 search generations using a multiple search hill climbing search. Result sets A and B were the results of generation 1's configurations, under two slightly differing test conditions. Both used the same experiment hardware, physical conditions and experiment framework. Set A experienced 5 trials per configuration. Set B and its series experienced 10 trials per configuration. Network noise drastically affected generations 2 and 3; meaning it would be inappropriate to compare results between generation 1 and the later generations.

F.8.1 Initial Configuration Set Random Selection

The initial configuration sets were generated using the `java.util.Random` library class. Result set A used a seed of 1. Result set B used a seed of 2. The `random.nextInt()` method was used to select the indexes within each parameter's index range in order. An analysis of the class's linear congruential generator algorithm is shown in C.2.

F.8.2 Other Result Sets

Table F.16 shows percentage configuration value of the top results from generation 1's result sets and an overview of the configurations tested during the search. These are given to provide information on bias of the configuration coverage of the search.

Tables F.17 to F.20 show the metric and immunisation rate performance scores of the top configurations during generations 1 and 2.

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	0.5	1.5	0.8	2	14.000	0.458	0.750	11.104	0.763	0.653	61.650	2.919	3.525	B
2	0.66	0.5	0.9	1	13.000	0.671	1.000	11.729	1.221	0.754	54.400	3.000	3.475	B
3	0.5	2.5	0.9	2	14.000	0.800	1.000	11.794	0.828	1.687	55.500	0.602	0.400	A
4	0.5	3.5	0.8	2	14.000	0.000	0.000	11.117	1.003	1.502	67.950	1.263	1.425	B
5	0.66	2.0	0.9	3	14.000	0.400	0.000	12.019	0.950	1.501	59.900	2.033	3.100	A
6	0.75	0.5	0.9	4	13.000	0.748	1.000	12.483	1.237	0.990	58.950	3.637	3.250	B
7	0.66	0.5	0.8	1	14.000	0.400	0.000	12.029	1.060	0.520	62.200	1.972	2.625	B
8	1.0	1.0	0.9	1	14.000	0.458	0.750	12.329	0.942	0.966	61.600	6.471	9.975	B
9	0.5	1.5	0.8	4	13.000	0.748	1.000	12.457	1.394	0.848	65.100	2.020	3.800	A
10	0.5	1.0	0.7	1	14.000	0.400	0.000	12.004	1.458	2.156	64.900	1.212	1.600	A

Table F.17 – Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result sets A and B.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	0.5	1.5	0.8	2	-1.51175	0.12884	0.12350	-2.22333	0.15673	0.25088	-1.58505	0.13073	0.13225
2	0.66	0.5	0.9	1	-1.62992	0.19850	0.13338	-2.22892	0.19823	0.13925	-1.68752	0.19766	0.14261
3	0.5	2.5	0.9	2	-1.64067	0.12630	0.24683	-2.25067	0.13546	0.25883	-1.70167	0.12719	0.24803
4	0.5	3.5	0.8	2	-1.50283	0.16717	0.25033	-2.35983	0.18162	0.26033	-1.58853	0.16850	0.25133
5	0.66	2.0	0.9	3	-1.65317	0.16403	0.26300	-2.38317	0.18819	0.34917	-1.72617	0.16616	0.27380
6	0.75	0.5	0.9	4	-1.73508	0.21127	0.18308	-2.41167	0.25505	0.27317	-1.80178	0.21494	0.18873
7	0.66	0.5	0.8	1	-1.66725	0.17626	0.09287	-2.42783	0.18828	0.15017	-1.74203	0.17710	0.09842
8	1.0	1.0	0.9	1	-1.70475	0.16151	0.18608	-2.46375	0.20429	0.20404	-1.77285	0.16169	0.20078
9	0.5	1.5	0.8	4	-1.72617	0.23676	0.11633	-2.46817	0.24625	0.16433	-1.80037	0.23742	0.12113
10	0.5	1.0	0.7	1	-1.65067	0.24053	0.33433	-2.47067	0.22600	0.35433	-1.73267	0.23901	0.33633

Table F.18 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result sets A and B.

Rank	Configuration Set				Metric Scores									Generation
					Distributed-M1			*Time-M2*			*DataSent-M3*			
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	
1	0.9	3.5	0.7	5	14.000	0.000	0.000	15.854	0.883	1.053	166.800	4.340	6.375	B_2
2	0.5	1.0	0.6	5	14.000	0.000	0.000	15.816	0.864	0.909	169.400	4.192	5.675	B_2
3	0.5	3.5	0.2	2	14.000	0.000	0.000	15.660	0.748	0.593	169.750	5.203	4.550	B_2
4	0.5	0.5	0.7	1	14.000	0.806	0.750	16.453	1.450	2.042	168.400	4.570	5.250	B_2
5	0.75	0.5	0.8	5	14.000	0.300	0.000	16.120	1.423	0.822	168.750	3.849	3.075	B_2
6	0.5	0.5	0.4	1	14.000	0.000	0.000	16.132	0.885	0.729	167.400	5.351	7.950	B_2
7	0.5	1.0	0.6	1	14.000	0.400	0.000	15.963	0.825	1.317	170.800	6.383	7.625	B_2
8	0.5	0.5	0.3	2	14.000	0.300	0.000	15.985	1.055	1.672	167.600	2.450	3.725	B_2
9	0.9	4.0	0.7	4	14.000	0.300	0.000	15.962	0.783	0.688	168.400	3.291	4.825	B_2
10	0.9	4.0	0.9	5	14.000	0.000	0.000	16.062	1.216	1.226	168.350	3.098	4.925	B_2

Table F.19 – Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_2.

Rank	Configuration Set				Immunisation Rate Scores								
					O1			*O2*			*O3*		
	P0	*P1*	*P2*	*P3*	μ	*Std*	*IQR*	μ	*Std*	*IQR*	μ	*Std*	*IQR*
1	0.9	3.5	0.7	5	-2.29242	0.14716	0.17554	-5.05108	0.16988	0.24175	-2.56532	0.14731	0.18564
2	0.5	1.0	0.6	5	-2.28608	0.14397	0.15142	-5.13425	0.20550	0.22663	-2.56928	0.14915	0.16062
3	0.5	3.5	0.2	2	-2.26000	0.12460	0.09879	-5.13717	0.19333	0.22150	-2.55272	0.12937	0.09765
4	0.5	0.5	0.7	1	-2.42967	0.25201	0.34029	-5.14967	0.25310	0.38308	-2.70167	0.25063	0.35059
5	0.75	0.5	0.8	5	-2.34925	0.23658	0.13700	-5.16050	0.24815	0.22925	-2.62695	0.23664	0.14560
6	0.5	0.5	0.4	1	-2.33867	0.14748	0.12146	-5.18683	0.19791	0.22037	-2.61967	0.14987	0.12131
7	0.5	1.0	0.6	1	-2.31042	0.14321	0.23462	-5.18742	0.16949	0.28404	-2.60262	0.14094	0.22111
8	0.5	0.5	0.3	2	-2.31417	0.17826	0.29087	-5.19217	0.18997	0.22938	-2.60197	0.17886	0.28472
9	0.9	4.0	0.7	4	-2.31033	0.13655	0.11467	-5.19342	0.13919	0.21083	-2.59623	0.13538	0.12748
10	0.9	4.0	0.9	5	-2.32708	0.20261	0.20433	-5.20067	0.23638	0.24312	-2.60928	0.20539	0.20408

Table F.20 – The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_2.

Endnotes

[1]Broadly, T-cell lymphocytes mechanism theories for identifying viruses within infected biological cells describe a *training process* in the thalamus that discards new T-cells with receptors (TCRs) that have a high affinity for *self* amino acid strings and a virgin/mature T-cell *recognition process* that interacts and compare the amino acid strings (peptides) from other cells with their trained receptors. In the latter process a cell destruct (apoptosis) cytokine is released if a string sequence is recognised. Notably, this is also the cause of immunodeficiency diseases.

[2]As network service discovery tools Swimmer (Swimmer, 2006, p1332) noted ZeroConf (http://www.zeroconf.org) and Bonjour (http://developer.apple.com).

[3]Address space layout randomisation (ASLR) is now a standard feature of today's operating systems to reduce the probability of successful buffer overflow jump exploits. A jump (JMP) exploit can enable an attacker to run code in static areas of memory, known either by runtime pointer reference or by prior knowledge. According to Shacham et al. (2004) (Shacham *et al.* , 2004) Address space layout randomization (ASLR) makes these types of attack "extremely unlikely to succeed on 64-bit machines as the memory locations of functions are random" whereas for "32-bit systems ASLR provides little benefit since there are only 16 bits available for randomization, and they can be defeated by brute force in a matter of minutes".

[4]Mori's (1993) (Mori, 1993) autonomous fault tolerance architecture is based on control and coordination enabled by precise communication; the *data field* component indicates that all subsystems (nodes) broadcast their data, however only nodes registered to the *data field* listen and respond to these broadcasts.

[5]That is, *only* in the event that zero detectors are found on at least one machine (which the experiment design creates if no distribution occurs), for all other known detectors (read-in elsewhere on the network) the median experiment total time value is used. From this set of time durations, the median time value is presented as the *Time-M2* value for the given trial. This is, by design, always the case for the final node under our malicious user-agent design, as that node reads-in 0% malicious inputs, consequently creating zero detectors. Therefore, the NoNet *Time-M2* time results are exactly the duration of time taken, by the median node, to complete the experiment.

[6]Python's statistical SciPy library implementation of the Mann Whitney U performs a one-tailed test. This can be converted to two-tailed with $P \times 2$ according to the SciPy test documentation.

[7]Apache Hadoop's website: https://hadoop.apache.org/ . Summarising description from the website "The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models."

[8]TypeSafe AKKA.io framework's website: http://akka.io/ . Summarising description from the website: "Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM."

Glossary

CARDINAL-Vanilla Our implementation and extension of CARDINAL. A number of changes to Kim et al.'s Danger Theory-inspired abstract model are discussed and tested within the thesis including vital implementation details and its application domain.. 127, 128, 141, 149, 200–202

CARDINAL Is a Danger Theory-inspired artificial immune system architecture model using an internal multi-agent system to determine priority with peer-to-peer network connectivity designed to detect and respond to worm and email spamming attacks. First published in (Kim *et al.* , 2005).. 15, 16, 69, 70, 121, 122, 124–126, 128, 137, 139, 140, 151, 163, 200, 201

AIS Artificial Immune System. 14, 15, 41, 42, 44, 45, 58, 69, 72, 142, 144, 145, 163, 197–202, 248

ASLR Address space layout randomisation (ASLR) is now a standard feature of today's operating systems to reduce the probability of successful buffer overflow jump exploits. A jump (JMP) exploit can enable an attacker to run code in static areas of memory, known either by runtime pointer reference or by prior knowledge. According to Shacham et al. (2004) (Shacham *et al.* , 2004) Address space layout randomization (ASLR) makes these types of attack "extremely unlikely to succeed on 64-bit machines as the memory locations of functions are random" whereas for "32-bit systems ASLR provides little benefit since there are only 16 bits available for randomization, and they can be defeated by brute force in a matter of minutes".. 52

BPF Berkeley Packet Filter (BPF) is an efficient binary message parsing system used to rapidly separate (or extract) packet property data from a stream of packets. A protocol schema (a filter definition) is required to "filter" (collect) or "drop" a packet.. 125

CTL Cytotoxic T-Lymphocyte. 72

DC The dendritic cell is a cell type of the innate immune system found in humans. It is named after its dendrite-like appearence (many branches). It is generalised as an antigen presenting cell (APC) as it collects, carries and presents antigen from peripheral areas of the body (such as the tissue or gut) to lymphocyte cells, typically in the lymphatic organs.. 72, 79, 198

DCA Dendritic Cell Algorithm. 69

DT Danger Theory. 69

Enclave An enclave is a logical grouping of users, assets, systems and/or services that define or fulfil a functional behaviour within the system. Enclaves represent regions or "zones" that can be used to isolate certain functions in order to secure them more effectively. Assets may be any hardware device within the network. Systems may be a collection of devices. Services may be a collection of software services on one or more devices.. 18

Fingerprinting Fingerprinting is the process of collecting outputs from open ports on a system and mapping those to a device or software version. This is can be used to monitor system state. However it is commonly used by penetration testers to recognise which applications are running to thus test for exploits.. 21

HIDS Host-based Intrusion Detection System (HIDS) is a type of distributed application with a client on every machine on a network, each performing intrusion detection activities and sharing state/ detectors with each other. Such activities include monitoring for abnormal traffic and comparing incoming network traffic against known detectors.. 69, 200

HIS Human Immune System. 28

ICS An Industrial Control System (ICS) consists of controller(s) (such as a programmable logic controller), sensor(s) and actuators to fulfil an automation role, typically for repetitive industrial application.. 14, 15, 18, 21, 24, 91, 107, 165, 188, 196, 197, 199, 200, 205, 254

MAS Multi-Agent System. 69, 198

PLC A programmable logic controller is a single processor microcontroller that executes a single binary program and thoroughly tested to ensure it operates in critical real-time dependent circumstances. The controller is typically programmed using ladder logic (LAD) like programming languages by engineers. They will typically use sensor(s) and actuators to fulfil an automation role, such as repetitive industrial applications.. 13, 17, 21, 58, 59, 200, 211

SCADA Supervisory Control and Data Acquisition (SCADA) refers to an Industrial Control System (ICS) with monitoring and control processes. A SCADA network typically carry remote sensors or actuators (i.e. separated by large geographical distance from the control and monitoring interfaces) and this fact is often used to differentiate them from ICS networks.. 13–15, 17, 19, 21, 23, 24, 65, 81, 83, 91, 107, 134, 144, 148, 165, 174, 188, 196, 197, 199, 200

SHS Self-Healing System. 91

ssh Secure Shell (ssh) is an encrypted network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers.. 140, 216

zero-day A zero-day vulnerability specifies the day on which a vulnerability is known by a party, often not the author(s) of the affected code. For example, a zero-day attack will use detailed knowledge of a zero-day vulnerability to exploit the weakness in order to carry out an intended action. The 0-th day can refer to the date of the vulnerability or attack's recognition, depending on the terminology user.. 17, 22, 23

List of Figures

2.1 Barrier-based *Defence-in-Depth* architecture, 2009 (p62), from NIST 800-82 Rev 2 Standard. . . . 21
2.2 Example of the corporate and control network topology division of industrial networks (p57), from NIST 800-82 Rev 2 Standard. . . . 22
2.3 Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.2 and Table A.3. . . . 23
2.4 Automated chemical release and monitoring remote terminal unit (RTU) station. Powered by photovoltaic and long range communication via directional radio antenna. In a field on a hill. *Image by Dan Steele, 2012.* . . . 23
2.5 Diagram illustrates a sophisticated attack with many innocuous steps. Showing single source of information is insufficient to distinguish a malicious attack from a benign update. 26

4.1 Snapp et al.'s 1991 DIDS Architecture Diagram . . . 45
4.2 Forrest et al.'s 1994 negative selection algorithm . . . 46
4.3 Marmelstein et al.'s 1998 Architecture Diagram . . . 47
4.4 Okamoto et al.'s 2000 Overview of Agent Control Mechanisms . . . 48
4.5 Harmer et al.'s 2002 Agent Mechanisms . . . 49
4.6 Brumley et al.'s 2002 Architecture for self-healing process state . . . 51
4.7 Kim et al.'s 2005 Overview of CARDINAL (lymph-tissue) distributed architecture. Extended by Fu et al. 2007. . . . 53

5.1 Overview of CARDINAL-Vanilla. . . . 61
5.2 Engineered System Overview of CARDINAL-Vanilla architecture client per node. . . . 61
5.3 Tissue and Lymph Node processes in CARDINAL-Vanilla in a single node. Numbered processes are described in 5.4. Diagram illustrates input data, data representation, agents, environments, agent interaction types (indicated by arrow line types) and agent network transmission messages. Diagram formatted as Fig. 2 from *(Kim et al., 2005)* for comparisons. . . . 65
5.4 Flow control diagram of the main Tissue processes. . . . 68
5.5 Flow control diagram of the main Lymph Node processes. . . . 68
5.6 Summarised view of Input to Response from the classifier decision perspective. . . . 70
5.7 Danger Input Validation Process - Naïve T-cell Maturation and Differentiation with DC Interaction . . . 72
5.8 Diagram of the data structure representation for the moving window of accumulated danger. $t - P3$ is the earliest time block and next to be removed, t is the current and most recently added time block. . . . 73
5.9 Volume Selection Feedback Loop - Sickness Focus. T_{UA} is the under attack threshold. $P0$ is the under attack percentage volume to transmit. R_{NUA} is the percentage to send while not under attack. S_v is the recent danger level affected by damaging inputs. S_v is increased by S_d danger signal inputs, otherwise its value will decay over time. A_n is the input antigen or data input values, while S_* is the input data's label. . . . 74

5.10 Effector T-cell Feedback Loop upon Cell Interaction 76
5.11 Prioritisation of CTL Responders through Regulation. 77
5.12 DC Lifespan and Decay Rate 78

6.1 Diagram of thread behaviour and experiment procedure 93
6.2 Result of function $a(i, n)$, showing monotonic approximate linearity and close to even distribution of percentage values per user (node). 96
6.3 Actual effects of noise variation upon the user model's learning experience percentage in a five node network test. Shown are percentages (y-axis) of anomalous inputs reported at each input batch request during a training phase (150 batches; 15000 inputs). Line colours represent behaviour at each node. 99
6.4 Actual user model behaviour. White columns show the anomalous dataset line number indexes (instances) that were read-in, per node (horizontal strip). Black shows coverage of missed (not read-in) dataset instances. 100
6.5 Distance ratio output from $P(k, m, w_k, c_k)$ for MAX and MIN semantics for range $[1, ..100]$ from $r = 15$. 107
6.6 Real number mappings of base-2 logarithms of $[-10, ..9]$. 107

7.1 Diagram of distribution (transmission) components under test circled in red, within the CARDINAL model. 112
7.2 Network topology example of five peers on a single network segment and experiment infrastructure for the self-healing system test. *DB* is a runtime database used to store the detectors, a *log* file (per instance) collects data from that instance which is later parsed to calculate test measurements. 112
7.3 Flow diagram of CARDINAL's detector selection process used during each transmission event. 113
7.4 *Dist-M1* variation of Vanilla performance over 25 runs with **(a)** 5, **(b)** 10, **(c)** 15 and **(d)** 20 nodes. Y-axis shows quantity of detectors (M1) from 0 to 14 (max), X-axis shows the test run number from 1 to 25. 119
7.5 Box plot observations of detector distribution quantity metric and multi-objective evaluation (O2) on the virtual network tests. 125
7.6 *Dist-M1* variation of Vanilla performance over 10 runs with **(a)** 5, **(b)** 10, **(c)** 15, **(d)** 20, **(e)** 30 and **(f)** 41 nodes. Y-axis shows quantity of detectors (M1) from 0 to 14 (max), X-axis shows the test run number from 1 to 10. 132
7.7 Box plot showing the relative difference in *Time-M2* metric scores for the Artificial Immune System (AIS) and engineered algorithms for selection and distribution of detector modules. 133
7.8 Box plot showing the relative difference in *DataSent-M3* metric scores for the AIS and engineered algorithms for selection and distribution of detector modules measured over each trial. 134
7.9 Box plot observations of detector distribution quantity metric and multi-objective evaluation (O2) on the enterprise network tests. 139
7.10 Sequence diagrams showing execution of experiment phase events w.r.t time. Time and ordered node numbers are plotted on Y-axis and X-axis. Black blocks show the Experiment Phases (including start and finish states) stated in 6.3 running on each node. *Note the vertical-dashed lines on* **(d)** *are an image scaling effect.* 142
7.11 Virtual network benchmark tests with 20 (0-19) nodes over 25 runs. 450ms time delay between each node start time. Plots from left to right show findings from algorithms: CARDINAL-Vanilla, Optimal-Static, Send-All. 149
7.12 Enterprise network benchmark tests with 41 nodes (0-40) over 10 runs. No time delay between node start times. Plots from left to right show findings from algorithms: CARDINAL-Vanilla, Optimal-Static. 149

7.13 Plots show time in seconds (Y-axis) for the most common detector to be received via network at each numbered node (X-axis) (in starting order). Time is relative from earliest receipt time. Virtual network benchmark tests with 20 (0-19) nodes over 20 runs running CARDINAL-Vanilla. Plots from left to right show findings from time delays: None, 0.45 s and 0.9 s. 150

8.1 Box plots showing metric performance while varying the under attack volume parameter (P0) value within the discrete testing range. 160
8.2 Box plots showing metric performance while varying the initial priority multiplier parameter (P1) value within the discrete testing range. 161
8.3 Box plots showing metric performance while varying the priority suppression multiplier parameter (P2) value within the discrete testing range. 162
8.4 Box plots showing metric performance while varying the static moving window size parameter (P3) value within the discrete testing range. 162
8.5 Figure shows median results from our three system performance measures O1, O2 and O3 (scored on the *Y-axis*) from 10 trials with each parameter's set of configuration values (*X-axis*). Greater scores values are preferred. The plots from left to right show parameters: P0, P1, P2, P3 and P4, each are described within the chapter. Default parameter configuration values were $[0.75, 2.0, 0.95, 60, 50]$ respectively. 167
8.6 Diagram showing the optimisation pipeline employed to tune the parameter configuration. 170
8.7 Illustration of the search states analysed within the text. Highlights the differing immunisation rates and evaluation equations used during the analysis. Green circles represent the configuration set selected during the step-wise search. Grey circles represent the configuration set selected post-search, using the Ratio of Distances evaluation equations. Four lines from each circle represent values along the four parameter dimensions. 172
8.8 The plot shows the median immunisation rates (O1,O2,O3) from the table above over the 10 search generations. 174
8.9 Diagram showing the optimisation pipeline employed in the parameter configuration search under the enterprise network test set-up. 179
8.10 Illustration of the search steps taken by the multi-start hill climbing parameter tuning approach. 179
8.11 The plot shows the median immunisation rates (O1,O2,O3) over the 3 search generations and highlights the experiment noise. 181
8.12 An illustration of the saturated regions of highest immunisation rate scores found within the parameter ranges for $P2$: [0.1-0.9] and $P3$: [1-5]. Circled are the regions preferred by the virtual and enterprise network conditions. 186

9.1 A toy-example topology with added hardware integrated and automation sensors numbered. 194
9.2 An example of default objective weightings assigned to each indicator category per device. 195
9.3 An example of the current indicator states as viewed from each device, updated by periodic state message transmissions. X'es are values unavailable to that device. 196
9.4 Diagram of intrusive social sensing behaviours as depicted by one-way requests (white arrows) and two-way requests (black arrows). Grey tables at intensive role (IR) devices show the collected local perspective knowledge of the other devices' normal behaviours. 200

A.1 Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.2. 213
A.2 Vulnerabilities per PLC manufacturer, ordered by total reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* Raw data in Table A.3. 214

A.3 Table shows compatibility of controller functions (first column) remotely called on the specified Siemens controllers (top row) as invoked via the Siemens S7 networked communication protocol. 215

B.1 Cellular signalling types. 217

C.1 Figure showing 32 images produced by java.util.Random's use of linear congruential algorithm with default seed and limit on nextInt(). 1 image for each of the 32 bit values of integers, using two's complement (signed bit is index 0) with big endian representation (8th bit in 4th byte represents the smallest number). Black and white represent 0 and 1 (non-zero). Rows 1 to 4 shows bits 0-7, 8-15, 16-23 and 24-31. Images are scaled down from 128 by 128 pixels over 128^2 bits. 221

E.1 Time taken to transmit the median detector. 229
E.2 Box plots observations of metric results from the virtual network tests. 230
E.3 Box plot observations of our Self-Healing System research objective evaluations (O1 and O3) as tested on a virtual network. Using our extension of the Weisbin & Rodriguez ratio equation, with the SendAll algorithm results used as reference. Greater values imply better performance. 231
E.4 Time taken to transmit the median detector. 232
E.5 Metric observations. 233
E.6 Box plot observations of our Self-Healing System research objective evaluations (O1 and O3) as tested on an enterprise network. Using our extension of the Weisbin & Rodriguez ratio equation, with the SendAll algorithm results used as reference. Greater values imply better performance. Note that plot y-axes are truncated to -1.5, in order to exclude Optimal-Static's outlier value transposed from the M2 (Time) metric results. . . 234

F.1 Under Attack Volume Percentage (P0) - parameter value range effects. 243
F.2 Initial Priority Value Multiplier (P1) - parameter value range effects. 243
F.3 Priority Value Suppression (P2) - parameter value range effects. 244
F.4 Static Moving Window Size (P3) - parameter value range effects. 244
F.5 Dendritic Cell Lifespan (P4) - parameter value range effects. 245
F.6 The box-plots show results of our three system performance measures (scored on the *Y-axis*) from trials with our five parameter's sets of configuration values (*X-axis*). Greater *Y-axis* values are preferred. The red dotted line shows the median trend. From left to right, P0, P1, P2, P3 and P4 parameter results are shown. Each are described within the chapter. 248

List of Tables

5.1 Table of important terms, their summarised definitions from biological and computational perspectives, their structure representation and usage within the architecture. A complete list of parameter mappings are specified in Table 5.2. 62
5.2 Table of CARDINAL-Vanilla parameters. Cell types parameters are described in text. Cells types are abbreviated as DC, TCN, CTL, Th2, Th1 in the table. Parameters are abbreviated as follows: A_* and S_* refer to data inputs, C_* refers to cytokine signalling, T_* refers to a threshold, V_* refers to an initial value, R_* refers to a rate, RV_* refers to a runtime variable and $P*$ refers to the key CARDINAL parameters under investigation. . 84
5.3 Table of key differences between the CARDINAL abstract model (Kim et al. 2005) and our CARDINAL-Vanilla model. 85

6.1 Table lists the event states of inputs to detectors in the architecture. Abstract model locations are in *italics*. Events with an asterix (*) are logged within local log measurement databases, once per state per unique input. (c) is a transfer between agent environments and is not logged. 89
6.2 Summary table showing the experiment procedure version development of the distributed system test procedure. 94
6.3 Summary table showing the experiment constants at each node of the distributed system under test. 94
6.4 Summary table showing the user model parameters. Values with square brackets show ranges, those without show constant values. 96
6.5 The "full" CSIC dataset version converts each training and testing set into a corresponding CSV file. 102
6.6 The "full v0.5" CSIC dataset version converts each training and testing set into a corresponding CSV file. 102
6.7 The "v0.5" CSIC dataset version takes a uniform random sub-sampling of the "full" dataset and reduces the quantity of attributes to three descriptive HTTP protocol data properties. 103
6.8 The "v0.5.2" CSIC dataset version takes an ordered sub-sampling of the "full" dataset and reduces the quantity of attributes to three descriptive HTTP protocol data properties. 103
6.9 Table of real value weightings per metric for each of the 3 objectives. 104
6.10 Table of the semantic preferences per metric. MAX indicates a larger value is preferred over a smaller value. MIN indicates the reverse. 105
6.11 Summary table of the evaluation equations used to combine multiple metric scores into a combined system performance measure score. 105
6.12 Table of example data for reference and test systems (M_i). The quantity of metrics and conditions of weights (w) and semantic measure conditions (c) match our third objective (O3). The individual performance measures (k) can be thought of as M1, M2 and M3 metrics. Systems (M_i) can be thought of as alternative algorithms, where M_1 is a benchmark algorithm. 108

6.13 Table of results from our $P(k, m, w_k, c_k)$ performance measure function in Equation 6.16 for individual measure data from Table 6.12. Results are truncated at 3 decimal places. 109
6.14 Table of results from $s(m)$ score function in Equation 6.17. Results are truncated at 6 decimal places. 109

7.1 Tabular results of metrics M1,M2,M3 showing medians (η) and standard deviations (*Std*) (with n-1 denominator) over 25 iterations of the four algorithms. 121
7.2 Mann-Whitney U test results for CARDINAL vs Optimal-Static of Metrics M1,M2,M3 over 25 iterations with Cohen d's difference and effect size. 121
7.3 Tabular results of objectives O1,O2,O3 showing medians (η) and standard deviations (*Std*) (with n-1 denominator) over 25 iterations of the four algorithms. 122
7.4 Mann-Whitney U test results for CARDINAL vs Optimal-Static of Objectives O1,O2,O3 over 25 iterations with Cohen d's difference and effect size. 122
7.5 Tabular results of metrics M1,M2,M3 showing medians (η), interquartile ranges (*IQR*) (75%-25%) and standard deviations (*Std*) (with n-1 denominator) over 10 iterations of each algorithm. Values are rounded to 3 significant digits. 135
7.6 Mann-Whitney U test results for CARDINAL vs OptimalStatic of Metrics M1,M2,M3 over 10 iterations with Cohen d's difference and effect size. An *Invalid* value in the *P* column and a *Nan* in the *U* statistic column represents identical results on all iterations of both algorithms in the respective Mann-Whitney U test. 135
7.7 Tabular results of objectives O1,O2,O3 showing medians (η), interquartile ranges (*IQR*) (75%-25%) and standard deviations (*Std*) (with n-1 denominator) over 10 iterations of the Vanilla and Optimal-Static algorithms. Values are rounded to 3 significant digits. . 136
7.8 Mann-Whitney U test results for CARDINAL vs OptimalStatic of Objectives O1,O2,O3 over 10 iterations with Cohen d's difference and effect size. 136
7.9 An informal summary of sequence analysis results from informal observation of an exemplar run of CARDINAL-Vanilla under different conditions: (a), (b), (c) and (d). Observed categorical values are given in the ranges *(high,med,low,none)* and *(small,med,large,unknown)*.144
7.10 Table of total experiment runtimes measured in seconds of CARDINAL-Vanilla under different conditions: (a), (b), (c) and (d). η is *(Std)* are medians and standard deviations over 10 trial runs. 144
7.11 Summary of metric results of virtual network tests with varying starting time delays between nodes. Table shows 5 network sizes (n) and results of metrics *Distributed-M1*,*Time-M2*,*DataSent-M3* data over 20 iterations. η and *(Std)* show medians and sample standard deviations. The Kruskal-Wallis null hypothesis shows whether the mean ranks of the 3 tests are the same, critical $\alpha = 0.05$. 145
7.12 Mann-Whitney U two-tailed test upon No Delay and 0.45s Delay *Distributed-M1* results from CARDINAL-Vanilla, with adjusted critical α as 0.0167. Showing metric samples of a similar population. 146
7.13 Mann-Whitney U corrected two-tailed test for population difference of CARDINAL-Vanilla over 10 runs, comparing *Distributed-M1* results, with adjusted critical α as 0.0167. Sizes *30 and *41 compare the virtual *Distributed-M1* results at n=20 with the enterprise *Distributed-M1* results at sizes $n = 30$ and $n = 41$. **(a)** Shows results of 0.45s Delay and enterprise network tests. **(b)** Shows results of No Delay and enterprise network tests. 147
7.14 Tables show Spearman's (rho) rank correlation statistic and significance (p is the P-value) under different test conditions. Each test attempts to correlate ranked detector delivery times via network transmission per node with starting node indices. Critical $\alpha = 0.05$. **(a)** Table shows benchmark virtual network tests over 25 runs, 20 nodes. **(b)** Table shows benchmark enterprise network tests over 10 runs, 41 nodes. **(c)** Table shows time delay virtual network tests over 20 Runs, 20 nodes. 151

8.1 Table shows the discrete testing range of values per parameter, as originally defined in 5.12, and the constant default parameter value while other parameters are varied. 157
8.2 Table of metric descriptions and references used in the architecture parameter range tests. The states refer to the state of the representation of an input, as Table 6.1. Refer to 6.2 for the metric M* descriptions. Preference is our preferred metric value, i.e. lower (MIN) or higher (MAX) values and '-' unimportant. 158
8.3 Summarising table of Spearman's rank (monotonic) correlations between each parameter's increasing range of values to the median results of each metric listed. The critical alpha for each two-tailed test is 0.05. Significance markers are based upon the P-values in the earlier test tables. "Pos." refers to positive (increasing) correlation and "Neg." refers to a negative (decreasing) correlation. 159
8.4 Table shows the mean average ($\bar{x}$) and standard deviation (*Std.*) of (difference) impact upon each increase of parameter value. Preferred impacts are in bold. Minimum values on *Time-M2* and *DataSent-M3* metrics are preferred. Greatest values on other metrics are highlighted. 159
8.5 Best and worst configuration values ($v.$) per parameter ranked by O1 median (μ) score. 165
8.6 Best and worst configuration values ($v.$) per parameter ranked by O2 median (μ) score. 165
8.7 Best and worst configuration values ($v.$) per parameter ranked by O3 median (μ) score. 165
8.8 Table showing the significance of difference between best and default parameter value configurations. Results are shown for two-tailed corrected Mann-Whitney tests (U) with an $\alpha = 0.05$ with 9 degrees of freedom (df). 166
8.9 Table shows the discrete testing range of values per parameter, adapted as a result of the earlier tests upon the original ranges. First generation default parameter value are shown. 170
8.10 Table showing the best configuration sets ranked at runtime by their *O3* immunisation rate score over the ten search generations running under the virtual network testing environment. Greatest immunisation rate scores (O1,O2,O3) are preferred. Preferred result values are shown in bold. The median *Time-M2* values were runtime decision factors used to calculate O1,O2 and O3, these are reported. The standard deviations (*Std*) and interquartile ranges (*IQR*) for the *Time-M2* metric are missing as they were not extracted at runtime. 174
8.11 Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the virtual network environment set-up. 176
8.12 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the virtual network environment set-up. 176
8.13 Table shows the discrete testing range of values per parameter, adapted as a result of the earlier tests upon the original ranges. 178
8.14 Table of the multi-start hill climbing algorithm's parameter values. 180
8.15 Table shows mean metric scores per generation from the top 10 configuration results ranked by immunisation rate for low throughput networks (O2). The *Diff.* columns show the multiplier difference from generation 1, at 3 decimal places. 182
8.16 Table of descriptive statistics shows a clear difference between generations, based upon the number of *IO-Events*, corroborating the belief that network throttling caused the metric differences between the generations. The *IO-Events* metric quantifies the number of transmission events by the median host of the network, per trial; stats are shown over the entire range of trials per generation. 182
8.17 The tables map each of the elite configurations' parameter values into either the top or bottom regions of the parameter's range. The percentages show where those configuration values fell from each of the elite configurations per generation. 183

8.18 Table shows a comparison of best parameter values selected during each of the parameter tuning searches. Emboldened are the opposing findings for parameters $P2$ and $P3$ in the enterprise to the virtual parameter tuning tests. L,ML,M and H refer to low to high values within each parameter's test range. 184
8.19 Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_3. 185
8.20 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_3. 185

A.1 Standards for securing Industrial Control System (ICS) and electronic networks. Organisation acronyms are listed in A.1.1. 211
A.2 Vulnerabilities reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* . . . 212
A.3 Vulnerabilities per PLC manufacturer, ordered by total reported to OSVDB during 2007–2015. *Retrieved on Aug 25th 2015.* 214

C.1 Tabular representation of related Figure C.1, showing mean of frequency of 0 bit values (black) per bit index using seeds 0 to 31. Rows 1 to 4 show bits 0-7, 8-15, 16-23, 24-31. See text and figure description for details. 220
C.2 Table showing standard deviation of frequency of 0 bit values (black) per bit index, using seeds 0 to 31. 221

D.1 Summary table showing dataset size and time taken to evaluate state of art network security SCADA and vulnerability security datasets using the C4.5 decision tree algorithm with one instance on one node. Time is taken over 10 runs. 226

F.1 Table of results from our $P(k, m, w_k, c_k)$ performance measure function in Equation F.5 for individual measure data from Table 6.12. Results are truncated at 3 decimal places. 238
F.2 Table of results from $s(m)$ score function in Equation F.6. Results are truncated at 6 decimal places. Noticeable differences in O3 values for systems $M_4, ..M_7$ of equal weight and improved distance. 239
F.3 Table of Spearman's rank rho (monotonic) correlations between the P0 range values and the median results for each metric listed. 240
F.4 Table of Spearman's rank rho (monotonic) correlations between the P1 range values and the median results for each metric listed. 240
F.6 Table shows measured results upon the variation of only the value v setting for parameters P0, P1 and P2. Tests are conducted under a given virtual network experiment set-up with five virtual hosts. Median μ and sampled standard deviation ($Std.$) summary statistics are reported per metric. 241
F.7 Table shows measured results upon the variation of only the value v setting for parameters P3 and P4. Tests are conducted under a given virtual network experiment set-up with five virtual hosts. Median μ and sampled standard deviation ($Std.$) summary statistics are reported per metric. 242
F.5 Table of Spearman's rank rho (monotonic) correlations between the full P2 range values and the median results for each metric listed. The full range tested here is between 0.1 and 2.0. 246
F.8 Table of Spearman's rank rho (monotonic) correlations between the reduced P2 range of values and the median results for each metric listed. The tested range here is between 0.1 and 1.0, a more intuitive range. 246
F.9 Table of Spearman's rank rho (monotonic) correlations between the P3 range values and the median results for each metric listed. 246

F.10 Table of Spearman's rank *rho* (monotonic) correlations between the P4 range values and the median results for each metric listed. 247
F.11 Tables of system performance measure results taken from 10 repeated trials with changes to each parameter's configuration value ($v.$). Median (μ) and standard deviations (*Std.*) are shown for each of the three performance measure objectives O1, O2 and O3. Greatest median and smallest *Std.* values are preferred and have been emboldened. *P2-Full* has been segregated as this range of values show particularly poor performance, as discussed in Part 1. 249
F.12 Table showing metric scores of the best configurations ranked by immunisation rate O1 score. Scores from the parameter tuning search under the virtual network environment set-up. 250
F.13 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O1 score. Scores from the parameter tuning search under the virtual network environment set-up. 250
F.14 Table showing metric scores of the best configurations ranked by immunisation rate O3 score. Scores from the parameter tuning search under the virtual network environment set-up. 251
F.15 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O3 score. Scores from the parameter tuning search under the virtual network environment set-up. 251
F.16 The table maps each of the elite configurations' parameter values into either the top or bottom regions of the parameter's range. The percentages show where those configuration values fell from each of the elite configurations. Shown are result sets A , B and a combination of A, B, B_2 and B_3 result sets. This table is included to give further informative coverage of the differing A and B set behaviours and an overview of all of the tested configuration sets. 252
F.17 Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result sets A and B. 253
F.18 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result sets A and B. . . 253
F.19 Table showing metric scores of the best configurations ranked by immunisation rate O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_2. 254
F.20 The adjoining table to the above, showing immunisation rate scores of the same best configurations ranked by their O2 score. Scores from the parameter tuning search under the enterprise network environment set-up during generation 3, result set B_2. 254

References

Adhikari, Uttam, Pan, Shengyi, & Morris, Tommy. 2013 (May). *Mississippi State Uni - ICS Attack Dataset – Retrieved on 2013.10.02. http://bespin.ece.msstate.edu/wiki/index.php/ICS_Attack_Dataset*.

Aharoni, Mati, Frankovic, Igor, OGorman, Jim, *et al.* . 2009. *The Offensive Security Exploit-DB Project. Retrieved on 2013.07.18. https://exploit-db.com*.

Alden, K., Timmis, J., Andrews, P., Veiga-Fernandes, H., & Coles, M. 2016. Extending and Applying Spartan to Perform Temporal Sensitivity Analyses for Predicting Changes in Influential Biological Pathways in Computational Models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **PP**(99), 1–1.

Alden, Kieran, Read, Mark, Timmis, Jon, Andrews, Paul S., Veiga-Fernandes, Henrique, & Coles, Mark. 2013. Spartan: A Comprehensive Tool for Understanding Uncertainty in Simulations of Biological Systems. *PLoS Comput Biol*, **9**(2), 1–9.

Alhomoud, Adeeb, Munir, Rashid, Disso, Jules Pagna, Awan, Irfan, & Al-Dhelaan, Abdullah. 2011. Performance evaluation study of intrusion detection systems. *Procedia Computer Science*, **5**, 173–180.

Alter, Galit, Heckerman, David, Schneidewind, Arne, Fadda, Lena, Kadie, Carl M, Carlson, Jonathan M, Oniangue-Ndza, Cesar, Martin, Maureen, Li, Bin, Khakoo, Salim I, *et al.* . 2011. HIV-1 adaptation to NK-cell-mediated immune pressure. *Nature*, **476**(7358), 96–100.

Aubert, Arnaud, & Renault, Julien. 2008. Cytokines and Immune-Related Behaviors. *Pages 527–547 of:* Phelps, & Korneva (eds), *Cytokines and the Brain*, vol. Volume 6. Elsevier.

Baker, Monya. 2007. Adult cells reprogrammed to pluripotency, without tumors. *Nature Reports Stem Cells*, **124**.

Baldauf, Matthias, Dustdar, Schahram, & Rosenberg, Florian. 2007. A survey on context-aware systems. **2**, 263–277.

Beck, Gregory, & Habicht, Gail S. 1996. Immunity and the invertebrates. *Scientific American*, **275**(5), 60–66.

Beltrami, Antonio P, Barlucchi, Laura, Torella, Daniele, Baker, Mathue, Limana, Federica, Chimenti, Stefano, Kasahara, Hideko, Rota, Marcello, Musso, Ezio, Urbanek, Konrad, *et al.* . 2003. Adult cardiac stem cells are multipotent and support myocardial regeneration. *Cell*, **114**(6), 763–776.

Binde, Beth, McRee, Russ, & OConnor, Terrence J. 2011. Assessing outbound traffic to uncover advanced persistent threat. *SANS Institute. Whitepaper.*

Binsalleeh, Hamad, Ormerod, Thomas, Boukhtouta, Amine, Sinha, Prosenjit, Youssef, Amr, Debbabi, Mourad, & Wang, Lingyu. 2010. On the analysis of the zeus botnet crimeware toolkit. *Pages 31–38 of: Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on.* IEEE.

Blaze, Matt, Feigenbaum, Joan, Ioannidis, John, & Keromytis, Angelos. 1999. The Role of Trust Management in Distributed Systems Security. *Pages 185–210 of:* Vitek, Jan, & Jensen, Christian (eds), *Secure Internet Programming*. Lecture Notes in Computer Science, vol. 1603. Springer Berlin / Heidelberg. 10.1007/3-540-48749-2_8.

Brambilla, Manuele, Ferrante, Eliseo, Birattari, Mauro, & Dorigo, Marco. 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, **7**(1), 1–41.

Brinkmann, Volker, Reichard, Ulrike, Goosmann, Christian, Fauler, Beatrix, Uhlemann, Yvonne, Weiss, David S., Weinrauch, Yvette, & Zychlinsky, Arturo. 2004. Neutrophil Extracellular Traps Kill Bacteria. *Science*, **303**(5663), 1532–1535.

Brumley, David, Newsome, James, & Song, Dawn. 2007. *Sting: An End-to-End Self-Healing System for Definding against Internet Worms, Malware Detection*. Springer Publications. Chap. 7, pages 147–170.

Burnet, F. M. 1959. *The Clonal Selection Theory of Acquired Immunity*. Vanderbilt Univ. Press, Nashville, TN.

Censor-Hillel, K., & Shachnai, H. 2011. Fast information spreading in graphs with large weak conductance. *Pages 440–448 of: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM.

Cherry, Steven. 2010. *IEEE Spectrum: How Stuxnet Is Rewriting the Cyberterrorism Playbook. Retrieved on 2012.07.18. http://spectrum.ieee.org/podcast/telecom/security/how-stuxnet-is-rewriting-the-cyberterrorism-playbook*.

Coffey, Neil. 2013. *java.lang.Random falls "mainly in the planes". Retrieved on 2014.02.26. http://www.javamex.com/tutorials/random_numbers/lcg_planes.shtml*.

Cohen, I.R. 2000. *Tending Adam's garden: evolving the cognitive immune self*. Academic Press.

Cohen, Jacob. 1988. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Hillsdale, NJ: Lawrence Earlbaum Associates.

Coico, R., & Sunshine, G. 2009. *Immunology: a short course*. Wiley-Blackwell.

Crouse, Michael, & Fulp, Errin W. 2011. A moving target environment for computer configurations using genetic algorithms. *Pages 1–7 of: Configuration Analytics and Automation (SAFECONFIG), 2011 4th Symposium on*. IEEE.

Danziger, M., & de Lima Neto, F.B. 2010. A Hybrid Approach for IEEE 802.11 Intrusion Detection Based on AIS, MAS and Naïve Bayes. *Pages 201–204 of: Hybrid Intelligent Systems (HIS), 2010 10th International Conference on*. IEEE.

Darwin, Charles. 1872. *On the Origin of Species. 6th Ed.* New York: Atheneum.

De Lemos, Rogério, Giese, Holger, Müller, Hausi A, Shaw, Mary, Andersson, Jesper, Litoiu, Marin, Schmerl, Bradley, Tamura, Gabriel, Villegas, Norha M, Vogel, Thomas, *et al.* . 2013. Software engineering for self-adaptive systems: A second research roadmap. *Pages 1–32 of: Software Engineering for Self-Adaptive Systems II*. Springer.

DeLoach, Scott A. 2000. *agentMOM User Manual. Retrieved on 2012.07.23. http://macr.cis.ksu.edu/agentmom*.

D'haeseleer, P. 1996. An immunological approach to change detection: Theoretical results. *Pages 18–26 of: In Proceedings of 9th IEEE Computer Security Foundations Workshop*. IEEE.

D'haeseleer, P., Forrest, S., & Helman, P. 1996. An immunological approach to change detection: Algorithms, analysis and implications. *Pages 110–119 of: Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE.

Dhaeseleer, P., Forrest, S., & Helman, P. 1997. A distributed approach to anomaly detection. *Submitted to ACM Transactions on Information System Security*.

Elsadig, M., & Abdullah, A. 2010 (jan.). Biological Intrusion Prevention and Self-Healing Model for Network Security. *Pages 337 –342 of: Future Networks, 2010. ICFN '10. Second International Conference on*.

Elsadig, M., Abdullah, A., & Samir, B.B. 2010a (june). Immune multi agent system for intrusion prevention and self healing system implement a non-linear classification. *Pages 1 –6 of: Information Technology (ITSim), 2010 International Symposium in*, vol. 3.

Elsadig, M., Abdullah, A., & Samir, B.B. 2010b (feb.). Intrusion Prevention and self-healing algorithms inspired by danger theory. *Pages 843 –846 of: Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 5.

Elsadig, Muna, & Abdullah, Azween. 2009. Biological Inspired Intrusion Prevention and Self-healing System for Network Security Based on Danger Theory. *International Journal of Video & Image Processing and Network Security*, **9**(October).

Engelmann, Ilka, & Pujol, Nathalie. 2010. Innate immunity in C. elegans. *Pages 105–121 of: Invertebrate Immunity*. Springer.

Etalle, Sandro, Gregory, Clifford, Bolzoni, Damiano, & Zambon, Emmanuele. 2014 (September). *Security Matters - SilentDefense ICS Whitepaper Retrieved on 2015.04.15. http://www.secmatters.com/sites/www.secmatters.com/files/documents/whitepaper_ics_EU.pdf*.

Falliere, Nicolas, Liam, O Murchu, & Chien, Eric. 2011. *Symantec Security Response: W32.Stuxnet Dossier. Retrieved on 2012.07.16. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf*.

Fayyad, Usama M., & Irani, Keki B. 1993. Multi-interval discretization of continuousvalued attributes for classification learning. *Pages 1022–1027 of: Thirteenth International Joint Conference on Articial Intelligence*, vol. 2. Morgan Kaufmann Publishers.

Field, Andy, & Hole, Graham J. 2003. *How to design and report experiments*. Sage.

Fink, Glenn A, & Oehmen, Chris S. 2012. Final Report for Bio-Inspired Approaches to Moving-Target Defense Strategies.

Forrest, S. 2011. *Lisys Software. Retrieved on 2011.12.01. http://www.cs.unm.edu/~forrest/software.html*.

Forrest, Stephanie, & Hofmeyr, Steven. 2001. Engineering an immune system. *GRAFT-GEORGETOWN-*, **4**(5), 369–369.

Forrest, Stephanie, Perelson, Alan S., Allen, Lawrence, & Cherukuri, Rajesh. 1994. Self-Nonself Discrimination in a Computer. *Pages 202–212 of: In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press.

Frauwirth, Kenneth A, Thompson, Craig B, *et al.* . 2002. Activation and inhibition of lymphocytes by costimulation. *The Journal of clinical investigation*, **109**(109 (3)), 295–299.

Fu, Haidong, Yuan, Xiguo, & Wang, Na. 2007 (dec.). Multi-agents Artificial Immune System (MAAIS) Inspired by Danger Theory for Anomaly Detection. *Pages 570 –573 of: Computational Intelligence and Security Workshops, 2007. CISW 2007. International Conference on.*

Gärtner, Bernd. 2000. Pitfalls in computing with pseudorandom determinants. *Pages 148–155 of: Proceedings of the sixteenth annual symposium on Computational geometry.* ACM.

Ghosh, Debanjan, Sharman, Raj, Raghav Rao, H., & Upadhyaya, Shambhu. 2007. Self-healing systems - survey and synthesis. *"Decision Support Systems"*, **42**(4), 2164 – 2185. Decision Support Systems in Emerging Economies.

Greensmith, Julie, Aickelin, Uwe, & Twycross, Jamie. 2006. Articulation and Clarification of the Dendritic Cell Algorithm. *Pages 404–417 of:* Bersini, Hugues, & Carneiro, Jorge (eds), *Artificial Immune Systems.* Lecture Notes in Computer Science, vol. 4163. Springer Berlin / Heidelberg.

Gronthos, S, Brahim, J, Li, W, Fisher, LW, Cherman, N, Boyde, A, DenBesten, P, Robey, P Gehron, & Shi, S. 2002. Stem cell properties of human dental pulp stem cells. *Journal of dental research*, **81**(8), 531–535.

Gu, Feng, Feyereisl, Jan, Oates, Robert, Reps, Jenna, Greensmith, Julie, & Aickelin, Uwe. 2011. Quiet in class: classification, noise and the dendritic cell algorithm. *Pages 173–186 of: Artificial Immune Systems.* Springer.

Guo, Yanhui, & Wang, Cong. 2005 (march). Autonomous decentralized network security system. *Pages 279 – 282 of: Networking, Sensing and Control, 2005. Proceedings. 2005 IEEE.*

Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, & Witten, Ian H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, **11**(1).

Harmer, Paul K., Williams, Paul D., Gunsch, Gregg H., & Lamont, Gary B. 2002. An artificial immune system architecture for computer security applications. *IEEE Transactions on Evolutionary Computation*, **6**, 252–280.

Harmer, P.K. 2000. *A distributed agent architecture for a computer virus immune system.* Tech. rept. DTIC Document.

Hettich, Seth, & Bay, SD. 1999. The UCI KDD Archive [http://kdd. ics. uci. edu]. Irvine, CA: University of California. *Department of Information and Computer Science*, 152.

Hofmeyr, Steven A., & Forrest, Stephanie. 2000. Architecture for an Artificial Immune System. *Evolutionary Computation*, **4**(4), 443–473.

Hofmeyr, Steven Andrew. 1999 (May). *An Immunological Model of Distributed Detection and Its Application to Computer Security.* Ph.D. thesis, University of New Mexico.

Hong, Sugwon, & Lee, Seung-Jae. 2008. Challenges and Perspectives in Security Measures for the SCADA System. *In: Proc. 5th Myongji–Tsinghua University Joint Seminar on Protection & Automation.*

Horn, P. 2001. Autonomic Computing: IBM's Perspective on the State of Information Technology. *Computing Systems*, **15**(Jan), 1–40.

Huebscher, Markus C., & McCann, Julie A. 2008. A survey of autonomic computing degrees, models, and applications. *ACM Comput. Surv.*, **40**(3), 7:1–7:28.

ICS-CERT. 2010 (September). *ICSA-10-238-01B: Stuxnet Malware Mitigation (Update B), September 15, 2010. Revised:2014.01.08. Retrieved 2015.08.25.. https://ics-cert.us-cert.gov/advisories/ICSA-10-238-01B.*

ICS-CERT. 2011 (July). *ICS-ALERT-11-186-01: Siemens SIMATIC Controllers Password Protection Vulnerability. July 2011. Retrieved 2015.08.25.. https://ics-cert.us-cert.gov/alerts/ICS-ALERT-11-186-01*.

ICS-CERT. 2015 (August). *ICS-ALERT-15-225-01A: Rockwell Automation 1769-L18ER and A LOGIX5318ER Vulnerability. Retrieved 2015.08.25.. https://ics-cert.us-cert.gov/alerts/ICS-ALERT-15-225-01A*.

Igure, Vinay M, Laughter, Sean A, & Williams, Ronald D. 2006. Security Issues in SCADA Networks. *Computers & Security*, **25**(7), 498–506.

Janeway Jr., C.A. 1989. Approaching the asymptote? Evolution and revolution in immunology. *Page 1 of: Cold Spring Harbor symposia on quantitative biology*, vol. 54.

Jesty, Robert, & Williams, Gareth. 2011. Who invented vaccination? *Malta Medical Journal*, **23**(02), 29.

Kaminka, Gal A., & Tambe, Milind. 2000. Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*.

Kang, Seung-Hoon, Park, Keun-Young, Yoo, Sang-Guun, & Kim, Juho. 2011. DDoS avoidance strategy for service availability. *Cluster Computing*, 1–8. 10.1007/s10586-011-0185-4.

Kephart, J. O. 1994. A Biologically Inspired Immune System for Computers. *Pages 130–139 of:* Brooks, R. A., & Maes, P. (eds), *Artificial Life IV Proceedings of the Forthe Int. Workshop on the Synthesis and Simulation of Living Systems*. MIT Press.

Kephart, Jeffrey O., Sorkin, Gregory B., Swimmer, Morton, & White, Steve R. 1997. Blueprint for a Computer Immune System. *In: Virus Bulletin International Conference in San Francisco, California, October 1-3, 1997*.

Kephart, J.O., & Arnold, W.C. 1994. Automatic Extraction of Computer Virus Signatures. *Pages 179–194 of: in: Proceedings of the Forth International Virus Bulletin Conference*. Virus Bulletin, Ltd.

Kephart, J.O., & Chess, D.M. 2003. The Vision of Autonomic Computing. *Computer*, **36**(1), 41–50.

Kephart, J.O., & Sorkin, G.B. 1997 (Mar. 18). *Generic Disinfection of Programs Infected with a Computer Virus*. US Patent 5,613,002.

Keromytis, Angelos D. 2007. Characterizing Self-Healing Software Systems. *In: In Proceedings of the 4th International Conference on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS)*.

Kim, J., Wilson, W., Aickelin, U., & McLeod, J. 2005. Cooperative automated worm response and detection immune algorithm (CARDINAL) inspired by t-cell immunity and tolerance. *Pages 168–181 of: ICARIS'05*. Springer.

Knapp, Eric D. 2011. *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems*. Syngress.

Kouns, Jake, & Martin, Brian. 2015. *Open Source Vulnerability Database (OSVDB) Search Engine. Retrieved on 2015.08.25. http://osvdb.org*.

Kruskal, William H, & Wallis, W Allen. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, **47**(260), 583–621.

Lafferty, K J, & Cunningham, A J. 1975. A NEW ANALYSIS OF ALLOGENEIC INTERACTIONS. *Aust J Exp Biol Med*, **53**(1), 27–42.

Lamont, G.B., Marmelstein, R.E., & Van Veldhuizen, D.A. 1999. A distributed architecture for a self-adaptive computer virus immune system. *Pages 167–184 of: New ideas in optimization*. McGraw-Hill Ltd., UK.

Le Goues, Claire, Dewey-Vogt, Michael, Forrest, Stephanie, & Weimer, Westley. 2012. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. *Pages 3–13 of: Software Engineering (ICSE), 2012 34th International Conference on*. IEEE.

Lewis, P.R., Chandra, A., Parsons, S., Robinson, E., Glette, K., Bahsoon, R., Torresen, J., & Yao, Xin. 2011 (oct.). A Survey of Self-Awareness and Its Application in Computing Systems. *Pages 102 –107 of: Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*.

Liu, Huan, & Motoda, Hiroshi. 1998. *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media.

Luk, Chi-Keung, Cohn, Robert, Muth, Robert, Patil, Harish, Klauser, Artur, Lowney, Geoff, Wallace, Steven, Reddi, Vijay Janapa, & Hazelwood, Kim. 2011. *PIN: A Dynamic Binary Instrumentation Tool. Retrieved on 2012.07.27. http://pintool.org/*.

Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., & Hazelwood, K. 2005. Pin: Building customized program analysis tools with dynamic instrumentation. *Pages 190–200 of: ACM SIGPLAN Notices*, vol. 40. ACM.

M. D. McKay, R. J. Beckman, W. J. Conover. 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, **21**(2), 239–245.

Mahdavi, Kiarash, Harman, Mark, & Hierons, Robert Mark. 2003. A multiple hill climbing approach to software module clustering. *Pages 315–324 of: Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*. IEEE.

Mandia, Kevin. 2013 (Feb). *Mandiant, APT1: Exposing One of Chinas Cyber Espionage Units (Accessed 2013.03.11), http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf*.

Mann, H. B., & Whitney, D. R. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, **18**(1), 50–60.

Marmelstein, Robert E., Veldhuizen, David A. Van, & Lamont, Gary B. 1998. A Distributed Architecture for an Adaptive Computer Virus Immune System. *Pages 11–14 of: Proceedings of the 1998 IEEE International Conference on Systems*.

Marsaglia, George. 1968. Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences of the United States of America*, **61**(1), 25.

Marsaglia, George. 2003. Random number generators. *Journal of Modern Applied Statistical Methods*, **2**(1), 2–13.

Matherly, John. 2009. *SHODAN the computer search engine. Retrieved on 2015.07.15. https://www.shodan.io*.

Matzinger, P. 1994. Tolerance, danger, and the extended family. *Annu Rev Immunol*, **12**, 991–1045.

Matzinger, Polly. 2002. The Danger Model: A Renewed Sense of Self. *Science*, **296**(5566), 301–305.

McClure, Stuart, Gupta, Shanit, Dooley, Carric, Zaytsev, Vitaly, Chen, Xiao Bo, Kaspersky, Kris, Spohn, Michael, & Permeh, Ryan. 2010. Protecting Your Critical Assets: Lessons Learned from Operation Aurora. *Retrieved on 2015.08.25.* http://www.wired.com/images_blogs/threatlevel/2010/03/operationaurora_wp_0310_fnl.pdf. *McAfee Labs and McAfee Foundstone Professional Services. Whitepaper.*

McFall-Ngai, Margaret. 2007. Adaptive immunity: care for the community. *Nature*, **445**(7124), 153–153.

Meserve, Jeanne. 2007 (Sept). *CNN Report on Idaho National Labs: Aurora – "Staged cyber attack reveals vulnerability in power grid." Retrieved on 2014.08.15. https://www.youtube.com/watch?v=fJyWngDco3g http://cnn.com/2007/US/09/26/power.at.risk/.*

Meysenburg, Mark M, & Foster, James A. 1999. Random generator quality and GP performance. *Pages 1121–1126 of: Proceedings of the genetic and evolutionary computation conference*, vol. 2.

Miller, Barton, Hollingsworth, Jeff, *et al.* . 2012. *DynInst Analysis Software Tool. Retrieved on 2012.07.27. http://dyninst.org/.*

Moore, HD, *et al.* . 2009. *The Metasploit Project. Retrieved on 2012.03.14. http://www.metasploit.com.*

Mori, K. 1993. Autonomous decentralized systems: Concept, data field architecture and future trends. *Pages 28–34 of: Autonomous Decentralized Systems, 1993. Proceedings. ISADS 93., International Symposium on.* IEEE.

Morris, Thomas, Srivastava, Anurag, Reaves, Bradley, Gao, Wei, Pavurapu, Kalyan, & Reddi, Ram. 2011. A control system testbed to validate critical infrastructure protection concepts. *International Journal of Critical Infrastructure Protection*, **4**(2), 88–103.

Mosteller, Frederick, & Tukey, John W. 1968. Data analysis, including statistics. *In Handbook of Social Psychology, Addison-Wesley, Reading, MA,.*

Murphy, Kenneth M, Travers, Paul, Walport, Mark, *et al.* . 2012. *Janeway's Immunobiology.* Vol. 7. Garland Science New York, NY, USA.

Narasipura, Srinivas D, Wojciechowski, Joel C, Charles, Nichola, Liesveld, Jane L, & King, Michael R. 2008. P-Selectin–Coated Microtube for Enrichment of CD34+ Hematopoietic Stem and Progenitor Cells from Human Bone Marrow. *Clinical chemistry*, **54**(1), 77–85.

Nardella, Davide. 2013 (September). *SNAP7 – Communication suite for natively interfacing with Siemens S7 PLCs. Retrieved on 2014.11.07. http://snap7.sourceforge.net/.*

Newsome, J., & Song, D. 2005. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. *In: In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS 05).* San Diego, CA: Internet Society.

Newsome, J., Brumley, D., Song, D., & Pariente, M.R. 2005. *Sting: An end-to-end self-healing system for defending against zero-day worm attacks on commodity software.* Tech. rept. Carnegie Mellon University.

Newsome, J., Brumley, D., & Song, D. 2006. *Sting: An end-to-end self-healing system for defending against zero-day worm attacks.* Tech. rept. Technical Report CMU-CS-05-191, Carnegie Mellon University School of Computer Science.

Nguyen, Hai Thanh, Torrano-Gimenez, Carmen, Alvarez, Gonzalo, Petrović, Slobodan, & Franke, Katrin. 2011. Application of the generic feature selection measure in detection of web attacks. *Pages 25–32 of: Computational Intelligence in Security for Information Systems.* Springer.

NIH. 2015 (March). *What are adult stem cells?. In Stem Cell Information [World Wide Web site]. Bethesda, MD: National Institutes of Health, U.S. Department of Health and Human Services, 2015. Retrieved on 2015.04.15. http:// stemcells.nih.gov/ info/ basics/ pages/ basics4.aspx*.

Okamoto, T., & Ishida, Y. 1999. A distributed approach to computer virus detection and neutralization by autonomous and heterogeneous agents. *Pages 328 –331 of: Autonomous Decentralized Systems, 1999. Integration of Heterogeneous Systems. Proceedings. The Fourth International Symposium on.*

Okamoto, Takeshi, & Ishida, Yoshiteru. 2000. A distributed approach against computer viruses inspired by the immune system. *IEICE transactions on communications*, **83**(5), 908–915.

O'Leary, Jacqueline G, Goodarzi, Mahmoud, Drayton, Danielle L, & von Andrian, Ulrich H. 2006. T cell–and B cell–independent adaptive immunity mediated by natural killer cells. *Nature immunology*, **7**(5), 507–516.

ONS, UK. 2013. *United Kingdom National Accounts, The Blue Book, 2013 Edition.* Office of National Statistics, UK Statistics Authority.

ONS, UK. 2014. *United Kingdom National Accounts, The Blue Book, 2014 Edition.* Office of National Statistics, UK Statistics Authority.

Ou, Chung-Ming. 2012. Host-based intrusion detection systems adapted from agent-based artificial immune systems. *Neurocomputing*, **88**, 78–86.

Ou, Chung-Ming, & Ou, Chung-Ren. 2010. Agent-Based immunity for computer virus: abstraction from dendritic cell algorithm with danger theory. *Pages 670–678 of: Advances in Grid and Pervasive Computing.* Springer.

Ou, C.M., & Ou, CR. 2011. Immunity-Inspired Host-Based Intrusion Detection Systems. *Pages 283–286 of: Genetic and Evolutionary Computing (ICGEC), 2011 Fifth International Conference on.* IEEE.

Ou, C.M., Wang, Y.T., & Ou, CR. 2011a. Intrusion detection systems adapted from agent-based artificial immune systems. *Pages 115–122 of: Fuzzy Systems (FUZZ), 2011 IEEE International Conference on.* IEEE.

Ou, C.M., Wang, Y.T., & Ou, C. 2011b. Multiagent-based dendritic cell algorithm with applications in computer security. *Intelligent Information and Database Systems*, 466–475.

Owens, Nick, Timmis, Jon, Greensted, Andrew, & Tyrell, Andy. 2007. On Immune Inspired Homeostasis for Electronic Systems. *Pages 216–227 of:* de Castro, Leandro, Von Zuben, Fernando, & Knidel, Helder (eds), *Artificial Immune Systems.* Lecture Notes in Computer Science, vol. 4628. Springer Berlin / Heidelberg.

Perkins, Jeff H, Kim, Sunghun, Larsen, Sam, Amarasinghe, Saman, Bachrach, Jonathan, Carbin, Michael, Pacheco, Carlos, Sherwood, Frank, Sidiroglou, Stelios, Sullivan, Greg, *et al.* . 2009. Automatically patching errors in deployed software. *Pages 87–102 of: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles.* ACM.

Peterson, Dale. 2009. Quickdraw: Generating security log events for legacy SCADA and control system devices. *Pages 227–229 of: Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology.* IEEE.

Polack, Fiona AC. 2010. Self-organisation for survival in complex computer architectures. *Pages 66–83 of: Self-Organizing Architectures.* Springer.

Pollet, Jonathan. 2002. Developing a solid SCADA security strategy. *Pages 148–156 of: Sensors for Industry Conference, 2002. 2nd ISA/IEEE.* IEEE.

Psaier, Harald, & Dustdar, Schahram. 2011. A survey on self-healing systems: approaches and systems. *Computing*, **91**(1), 43–73. 10.1007/s00607-010-0107-y.

Quinlan, Ross. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

Ralston, PAS, Graham, JH, & Hieb, JL. 2007. Cyber Security Risk Assessment for SCADA and DCS Networks. *ISA transactions*, **46**(4), 583–594.

Rigoutsos, Isidore, & Floratos, Aris. 1998. Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics*, **14**(1), 55–67.

Rodriguez, Guillermo, & Weisbin, Charles R. 2003. A new method to evaluate human-robot system performance. *Autonomous Robots*, **14**(2-3), 165–178.

Roesch, Martin, *et al.* . 1999. Snort: Lightweight Intrusion Detection for Networks. *Pages 229–238 of: LISA*, vol. 99.

Rowstron, Antony, & Druschel, Peter. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Pages 329–350 of: Middleware 2001.* Springer.

Ryu, Dae Hyun, Kim, HyungJun, & Um, Keehong. 2009. Reducing security vulnerabilities for critical infrastructure. *Journal of Loss Prevention in the Process Industries*, **22**(6), 1020–1024.

Sangster, Benjamin, OConnor, T, Cook, Thomas, Fanelli, Robert, Dean, Erik, Adams, William J, Morrell, Chris, & Conti, Gregory. 2009. Toward instrumenting network warfare competitions to generate labeled datasets. *In: Proc. of the 2nd Workshop on Cyber Security Experimentation and Test (CSET09).*

SCMag. 2014 (April). *SCMagazine Product Review: AlienVault Unified Security Management v4.4 SIEM. Retrieved on 2015.09.16. http://www.scmagazine.com/alienvault-unified-security-management-v44/review/4143/*.

SCMag. 2015 (May). *SCMagazine Product Review: McAfee Enterprise Security Manager (ESM) SIEM. Retrieved on 2015.09.16. http://www.mcafee.com/uk/resources/reviews/sc-magazine-esm-5-star-rating.pdf*.

Scully, Peter, Song, Jingping, Disso, Jules Pagna, & Neal, Mark. 2013. CARDINAL-E: AIS Extensions to CARDINAL for Decentralised Self-Organisation for Network Security. *Pages 1235–1236 of: Advances in Artificial Life, ECAL*, vol. 12.

Shacham, H., Page, M., Pfaff, B., Goh, E.J., Modadugu, N., & Boneh, D. 2004. On the effectiveness of address-space randomization. *Pages 298–307 of: Proceedings of the 11th ACM conference on Computer and communications security.* ACM.

Shalem, Amir, Shemer, Eli, Keren, Guy, Ben-Yehuda, Muli, Agmon, Orna, Fish, Shlomi, Segall, Itai, Ben-Yossef, Gilad, & Friedman, Roy. 2003. *syscalltracker: Sourceforge Project.*

Silverstein, A.M. 2009. *A history of immunology.* Academic Press. Academic Press/Elsevier.

Snapp, S.R., Brentano, J., Dias, G., Goan, T., Granee, T., Heberlein, L.T., Ho, C.L., Levitt, K.N., Mukherjee, B., Mansur, D.L., *et al.* . 1991a. *Intrusion Detection Systems (IDS): A Survey of Existing Systems and a Proposed Distributed IDS Architecture.* Tech. rept. Technical Report CSE-91-7, Division of Computer Science, University of California, Davis.

Snapp, Steven R., Brentano, James, Dias, Gihan V., Goan, Terrance L., Heberlein, L. Todd, lin Ho, Che, Levitt, Karl N., Mukherjee, Biswanath, Smaha, Stephen E., Grance, Tim, Teal, Daniel M., & Mansur, Doug. 1991b. DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype. *Pages 167–176 of: In Proceedings of the 14th National Computer Security Conference.*

Somayaji, A., Hofmeyr, S., & Forrest, S. 1998. Principles of a computer immune system. *Pages 75–82 of: Proceedings of the 1997 workshop on New security paradigms.* ACM.

Sompayrac, L. 2003. *How the immune system works.* How It Works Series. Blackwell Pub.

Song, Jingping, Zhu, Zhiliang, Scully, Peter, & Price, Chris. 2013. Selecting Features for Anomaly Intrusion Detection: A Novel Method using Fuzzy C Means and Decision Tree Classification. *Pages 299–307 of: Cyberspace Safety and Security.* Springer.

Song, Jingping, Zhu, Zhiliang, Scully, Peter, & Price, Chris. 2014. Modified Mutual Information-based Feature Selection for Intrusion Detection Systems in Decision Tree Learning. *Journal of computers*, **9**(7), 1542–1546.

Spearman, Charles. 1904. The proof and measurement of association between two things. *The American journal of psychology*, **15**(1), 72–101.

Spencer, Herbert. 1896. *The principles of biology.* Vol. 1. D. Appleton.

Srinivasan, S.M., Kandula, S., Andrews, C.R., & Zhou, Y. 2004. Flashback: A lightweight extension for rollback and deterministic replay for software debugging. *Pages 3–3 of: Proceedings of the annual conference on USENIX Annual Technical Conference.* USENIX Association.

Stepney, Susan, Smith, Robert E., Timmis, Jonathan, Tyrrell, Andy M., Neal, Mark J., & Hone, Andrew N. W. 2005. Conceptual Frameworks for Artificial Immune Systems. *International Journal of Unconventional Computing*, **1**(3), 315–338.

Stibor, Thomas, Oates, Robert, Kendall, Graham, & Garibaldi, Jonathan M. 2009. Geometrical insights into the dendritic cell algorithm. *Pages 1275–1282 of: Proceedings of the 11th Annual conference on Genetic and evolutionary computation.* ACM.

Stoica, Ion, Morris, Robert, Karger, David, Kaashoek, M Frans, & Balakrishnan, Hari. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, **31**(4), 149–160.

Stouffer, Keith, Pillitteri, Victoria, Lightman, Suzanne, Abrams, Marshall, & Hahn, Adam. 2015. Guide to industrial control systems (ICS) Security. NIST 800-82 Rev.2. *NIST Special Publication*, **1**(800-82), 247.

Swimmer, M. 2006. Using the Danger Model of Immune Systems for Distributed Defense in Modern Data Networks. *Computer Networks*, **51**(5), 1315–1333. Elsevier.

Terunuma, Hiroshi, Deng, Xuewen, Dewan, Zahidunnabi, Fujimoto, Shigeyoshi, & Yamamoto, Naoki. 2008. Potential role of NK cells in the induction of immune responses: implications for NK cell–based immunotherapy for cancers and viral infections. *International reviews of immunology*, **27**(3), 93–110.

Torrano-Giménez, Camen, Perez-Villegas, Alejandro, Álvarez Marañón, Gonzalo, *et al.* . 2010a. *An Anomaly-Based Approach for Intrusion Detection in Web Traffic.* Tech. rept.

Torrano-Giménez, Carmen, Perez-Villegas, Alejandro, & Álvarez Marañón, Gonzalo. 2010b. An anomaly-based approach for intrusion detection in web traffic. *Journal of Information Assurance and Security*, **5**(4), 446–454.

Torrano-Gimenez, Carmen, Nguyen, Hai Thanh, Alvarez, Gonzalo, Petrovic, Slobodan, & Franke, Katrin. 2011. Applying feature selection to payload-based Web Application Firewalls. *Pages 75–81 of: Security and Communication Networks (IWSCN), 2011 Third International Workshop on.* IEEE.

Travers, Paul, Walport, Mark, & Janeway, Charles. 2008. *Janeway's immunobiology.* Vol. 978. Garland Pub.

Tsang, Rose. 2010. Cyberthreats, Vulnerabilities and Attacks on SCADA Networks. *University of California, Berkeley, Working Paper, http://gspp. dreamhosters. com/iths/Tsang_SCADA% 20Attacks. pdf (as of Mar. 25, 2013).*

U.S. Department of Homeland Security. 2009 (October). *Recommended Practice: Improving Industrial Control Systems Cybersecurity with Defense-in-Depth Strategies. October 2009. Retrieved on 2015.04.16. https://ics-cert.uscert.gov/sites/default/files/recommended_practices/Defense_in_Depth_Oct09.pdf.*

Vivier, Eric, Raulet, David H, Moretta, Alessandro, Caligiuri, Michael A, Zitvogel, Laurence, Lanier, Lewis L, Yokoyama, Wayne M, & Ugolini, Sophie. 2011. Innate or adaptive immunity? The example of natural killer cells. *Science*, **331**(6013), 44–49.

White, Tom. 2009. *Hadoop: The Definitive Guide.* 1st edn. O'Reilly Media, Inc. Apache Hadoop.

Wiens, Thomas. 2013. *S7comm Wireshark Dissector Plugin. Retrieved on 2014.11.08. https://wiki.wireshark.org/S7comm.*

Yeo, M., *et al.* . 2010 (Aug. 31). *Extrusion detection using taint analysis. US Patent 7,788,235.* Filed 2006.

Zaharia, Matei, Chowdhury, Mosharaf, Franklin, Michael J, Shenker, Scott, & Stoica, Ion. 2010. Spark: cluster computing with working sets. *Page 10 of: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10. Apache Spark.

Zhu, Bonnie, & Sastry, Shankar. 2010. SCADA-specific Intrusion Detection/Prevention Systems: A Survey and Taxonomy. *In: Proceedings of the 1st Workshop on Secure Control Systems (SCS).*

Zuk, Patricia A, Zhu, Min, Ashjian, Peter, De Ugarte, Daniel A, Huang, Jerry I, Mizuno, Hiroshi, Alfonso, Zeni C, Fraser, John K, Benhaim, Prosper, & Hedrick, Marc H. 2002. Human adipose tissue is a source of multipotent stem cells. *Molecular biology of the cell*, **13**(12), 4279–4295.