NETWORK AIS-BASED DDOS ATTACK DETECTION IN SDN ENVIRONMENTS WITH NS-3

A Thesis

Submitted to the Faculty

of

Purdue University

by

Stefan G. Jevtic

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electric and Computer Engineering

August 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Dongsoo Kim

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Xiao Luo

Department of Computer Information Technology

Approved by:

Dr. Brian King

Head of Departmental Graduate Program

This thesis work is dedicated to Jessica McClintic, who has been my spiritual nourishment through the challenges of graduate school and broader life. I am truly lucky and thankful for having you in my life. This work is also dedicated to my parents, Goran and Elaine Jevtic, who taught me that the highest satisfaction can only come from pushing my own limits.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Dongsoo Kim and my thesis committee members Dr. Brian King and Dr. Xiao Luo of the Purdue School of Engineering and Technology at Indiana University Purdue University Indianapolis for their advice and guidance. Special thanks are extended to Dr. Kim for his tutelage in the ways of engineering and problem solving, and Sherrie Tucker, the salt of the earth without whom little of this work would have born fruit. I would also like to thank the ECE Department faculty for providing an academic environment that allowed me to grow into what I am today.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

Jevtic, Stefan G. MSECE., Purdue University, August 2017. Network AIS-based DDoS Attack Detection in SDN Environments with NS-3. Major Professor: Dongsoo Kim.

With the ever increasing connectivity of and dependency on modern computing systems, our civilization is becoming ever more susceptible to cyberattack. To combat this, identifying and disrupting malicious traffic without human intervention becomes essential to protecting our most important systems. To accomplish this, three main tasks for an effective intrusion detection system have been identified: monitor network traffic, categorize and identify anomalous behavior in near real time, and take appropriate action against the identified threat. This system leverages distributed SDN architecture and the principles of Artificial Immune Systems and Self-Organizing Maps to build a network-based intrusion detection system capable of detecting and terminating DDoS attacks in progress.

1. INTRODUCTION

The size of the internet sphere has seen explosive growth in the last 10 years, and the rate of growth is expected to accelerate into the near future [1]. This accelerated growth is attributed to two main factors: the rise of the Internet of Things (IoT) and the emergence of cloud computing. The Internet of Things is the term attached to the growing network of small embedded devices that frequently serve the automation and security needs of homes and businesses. Designed to send and receive data with remote logging servers, or to be remotely controllable by a user's smartphone or web application software, these devices communicate over IP networks with or without requiring a human to be in the loop. While this offers increased consumer convenience and control, it also opens the door to new avenues of cyberattack, as the domain of potential targets grows and there is less human monitorization.

Much of this growth is attributed to consumer "Internet Connected Things" which include household appliances, security cameras, televisions, digital video recorders, and smart electric meters [1]. These devices are broadly connected to the internet through normal IP networks and thus can send/receive traffic to/from any device reachable from the internet. As of 2017, few of these devices possess adequate security measures, and thus are subject to infiltration [2]. Infiltrated or compromised devices can continue to provide expected consumer function and generally do not give any indication to the consumer that they have been compromised. Since these devices have the ability to connect to any remote internet endpoint and are very unlikely to be discovered once they are, internet connected things perfect candidates for inclusion into networks of malware infected devices (botnets).

At the other end of the spectrum, cloud computing has evolved as a resource sharing computing model in which large service providers provide infrastructre (Infrastructure As A Service), platforms, (Platform As A Service), software (Software

As A Service), and everything in between as leasable services. These services are profitable for the service providers once a sufficient level of resource utilization is reached, and this gives rise to multiple customers sharing resource space at the CPU, storage, and network levels. This is known as multi-tenancy. This availability has driven a centralization of enterprise class computing into the cloud computing model, which offers convenience and quality for businesses. However, this also entails new challenges as cloud providers contend with the technical and security issues of multi-tenancy, which open up novel cyberattack methods as potential attackers may now reside within the same server or even the same physical CPU as their intended victims.

A common and problematic cyberattack is the denial-of-service attack. It is characterized by an attempt by a remote device to send more traffic to a target system than that system is able to handle. Once this threshold capacity is reached, normal/ benign traffic can no longer be serviced, and the target becomes unavailable for use. The security community has been able to create effective firewall and filtering techniques to mitigate such DoS attacks, a state of affairs which then caused the attacks to evolve into distributed-denial-of-service (DDoS) attacks. DDoS attacks achieve the same end as their DoS ancestors, but do so by overwhelming the target with requests from a myriad of source devices called bots. In this way, the true source of the attack is very difficult to detect, as each attacking device does not send enough individual traffic to raise firewall/ filtering alarms. The effect, however, is the same: the summation of all the botnet traffic is enough to render the target resource unavailable. Effective protection against DDoS attacks remains an open problem in the security community.

To contend with the technical challenges of resource sharing, most cloud providers have developed concepts of software-defined networking (SDN) and network virtualization. In contrast with previous packet or circuit switching networks, software defined networks allow for software management of network traffic. This is enabled by centrally gathering network traffic information from a management network of distributed devices capable of monitoring and reporting traffic, then making intelligent decisions that lead to the highest resource utilization in the given situation. Because these networking models are associated with increased traffic metadata, they are well suited to tackling the problems of cyberattack [3].

This project proposes a system for recognizing and mitigating DDoS attacks in progress. The system's distributed detection and intervention strategies are bio-inspired and take inspiration from the function of human immune systems in detecting and disrupting unknown pathogens. Like its biological counterpart, this AIS is designed to be highly adaptive by using the distributed monitoring approach afforded by SDN systems with the malicious pattern generation techniques of a modified artificial immune system. The system is shown to be capable of being highly effective in monitoring, detecting, and mitigating unknown DDoS attacks in progress.

1.1 Challenges and Hypothesis

The primary challenges faced in the design of a system effective in mitigating real world DDoS attack are:

1. Monitor network traffic to level sufficient for the security needs of the specific application.
2. Detect attack behavior before or when it occurs.
3. Intervene and take most appropriate action against detected threat.

This work hypothesizes that an SDN environment with OFv1.3 switches is capable of monitoring a cloud environment's network to a level sufficient to enable an SOM based AIS model to detect a DDoS attack in progress. Upon successful detection, the AIS can define a new network policy capable of mitigating the attack and then instruct the OpenFlow network controller to enforce the attack mitigation policy.

1.2 Proposed System

This work proposes a network-based [] SOM-modified-AIS OpenFlow network application to meet the primary challenges of effective DDoS detection and mitigation in cloud computing environments. The system features collaborative profiling to meet the distributed monitorization needs of such environments, a time-series toxicity “overwhelm-factor” to model the effect a DDoS attack has on network terminal equipment, a network capable of simulating a DDoS attack in an SDN network environment, and a SOM-modified AIS network application to handle attack detection and the immune response. As a proof of concept, this work implements the SDN DDoS network, collaborative profiling monitorization scheme, and toxicity modeling AIS application while assuming a functional SOM block.

2. BACKGROUND

2.1 Denial of Service Attacks

Denial-of-service attacks target an end server by overwhelming its ability to service application level IP protocol requests. These DDoS attacks can feature a botnet composed of thousands of bots, each sending just a few HTTP requests per second. These are not limited to small servers, or a low number of requests, as was the case in the attack on an Incapsula researcher's website host in late 2016, which saw a peak of 35,000 requests per second [4]. Given that most midsized websites have a processing capacity in the 100 requests per second area, even a modestly sized botnet can cause large problems.

2.1.1 DDoS Attack Types

DDoS attacks take different forms, and the basic types are now introduced. Modern attacks may not necessarily fall into a single category, but can take elements of each to develop new types of DDoS attacks.

UDP Flood

User Datagram Protocol (UDP) flooding attacks occur when a botnet overwhelms the target by sending an IP packets of UDP datagrams at an unserviceable rate [5]. Since UDP transport sockets may be installed on a variety of ports, the target port is usually random and thus difficult to predict. The behavior of the target is to receive the UDP datagram, unpack it, and check for an associated application on the datagram's destination transport port. After it fails to find such an application, a destination unreachable Internet Control Message Protocol (ICMP) packet is generated and sent back to the attacker. This process of reception, check, and response takes non-trivial time and compute resources, leading to the resource becoming available if its processing capacity threshold is crossed.

TCP SYN Flood

Transmission Control Protocol (TCP) SYN flooding attacks are the result of a manipulation and understanding of the TCP three-way-handshake connection initialization process. TCP is a connection-oriented protocol, and features an initial process to set up the connection. The connection initiator begins by sending a TCP-SYN request to the connection receiver. Upon reception of the TCP-SYN, the receiver opens an active port on its system, and usually spawns a new process dedicated to listening for incoming packets (called the Transmission Control Block (TCB)) [6], and sends a SYN-ACK back to the initiator, indicating the receiver is ready and listening. Upon receiving the SYN-ACK, the initiator sends an ACK message and the connection is established. This is called the three-way-handshake [7].

A TCP-SYN flood attack occurs when attackers flood the target system with TCP-SYN requests [8]. The target servers handle them in the normal way, dedicating a port, internal memory, and usually a CPU process to listening for the SYN-ACK. However, in a TCP-SYN flood attack, the SYN-ACK is never sent, and the target server is left hanging, listening for a message that will never come. This hanging persists until a clock timeout occurs, and the TCB is freed. During this hanging period, if enough TCP-SYN requests have flooded, the server will be unable to grant a new TCB to a legitimate request, and the server is seen as unresponsive. Given the more substantial target resources that respond to a TCP-SYN request, flooding occurs with fewer attack resources than a comparable UDP or ICMP flood [8].

ICMP (Ping) Flood

Ping floods are a simple type of DDoS attack which see the target flooded with an unserviceable rate of ICMP echo request packets. Upon reception of an ICMP echo request, the server reads the source IP address and sends an ICMP echo reply message to that address. These are relatively quick and easy for a server to generate and send, so Ping floods generally target a network's bandwidth rather than the server's processing capacity [9]. As is the case with the other types, once a capacity threshold is reached, the server becomes unresponsive as it cannot be reached due to network congestion.

ICMP (Smurf) Flood

Smurf floods are another ICMP based attack type. A smurf flood attack, however, is an indirect type of attack in which the server systems are flooded with ICMP echo replies rather than ICMP echo requests [9]. This is done when the attacker spoofs the IP source address in its initial ICMP echo request to be the IP address of its intended attack victim. The attacker can then broadcast these spoofed echo request messages to whole networks/sub-networks of legitimate devices. These legitimate devices then generate and send ICMP echo reply messages to the spoofed source IP address, which is the address of the intended victim, and an attack occurs. In this way a DDoS attack can actually originate from a single root source.

2.1.2 DDoS Detection and Mitigation

General approaches to protecting against and mitigating DDoS attacks are well established [10] and involve the tasks of prevention, detection and filtering, and source determination. Traditional destination based firewall and filtering methods exist and have achieved some success [11]. Destination-based methods involve the planting of a network monitoring resource at the point at which the network connects the protected destination. This method does not protect the network as a whole, and usually feature little to no learning capability. Further, destination-based methods are ill-suited to the challenges of DDoS in cloud computing environments, where it is prohibitively impractical to guard each destination individually. Instead, SDN principles allow for a central agent to have a global view of the network itself and thus allow for network-based [12] detection methods.

Network-based detection involves distributing a set of network monitors throughout the network. Each monitor contains attack signatures which the network monitor seeks to match against the traffic flowing through it. These network monitors operate similarly to wiretaps, allowing an agent to view the traffic without disrupting it. Traditional network-based [12] detection also features little to no learning capability, instead relying on external analyses to create appropriate attack signatures for the monitors.

2.2 Software Defined Networking

Traditional packet switched networks feature devices that must be manually configured when network changes occur, and cannot adapt on the fly to changes in topology, congestion, network attack status, or general network policy. These networks function, but are expensive to maintain and are not well-suited to operation in cloud computing environments. Software Defined Networking is an emerging networking paradigm that seeks to give network administrators remote control over network operations. Such is accomplished by separating computer networks into three disjoint planes of operation: data, control, and management.

The data plane is composed of the network data traffic a network is intended to carry. This data is to be intelligently forwarded according to L2 (data link layer) and L3 (network layer) routing protocols and does not contain any messages or commands intended for the network devices themselves. The data plane simply contains the network payload. The control plane is composed of the network traffic protocols that are used to populate the forwarding tables of the data plane devices. These are the L2 and L3 protocols that make forwarding decisions at each data plane device. The management plane contains the software tools, including Simple Network Management Protocol (SNMP) based tools, that are used to monitor and configure network devices [13].

The data and control planes are unified in traditional networks, where network forwarding devices such as switches and routers operate independently of each other. In these networks, it is up to the control configuration on each device to ensure that data plane traffic can proceed as intended. The result is a highly decentralized network that must be manually configured at the device level to support a given network topology, routing protocol, or network policy.

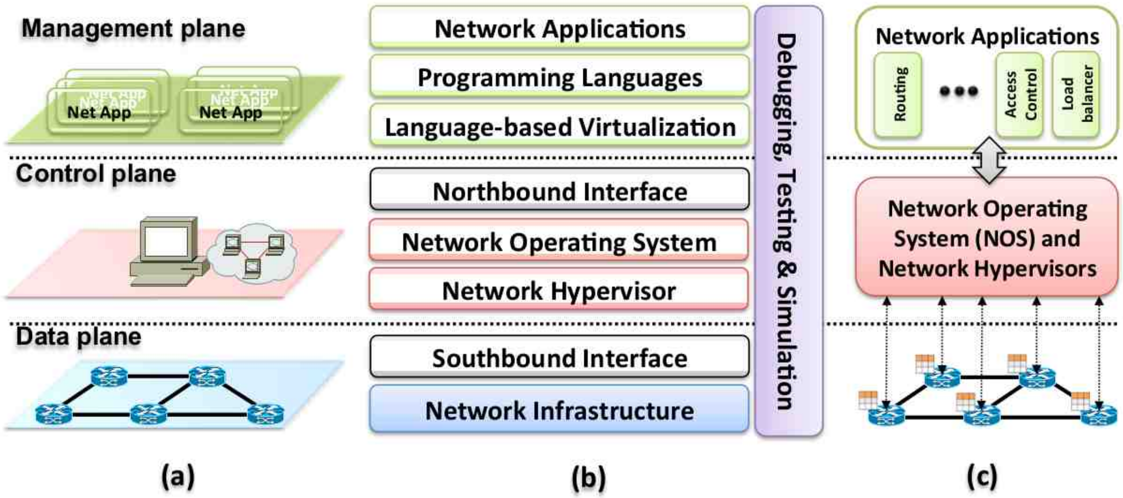

Fig. 2.1. Software-Defined Networks in (a) planes, (b) layers, (c) system design architecture [13].

By contrast, SDN is characterized by a separation of all three planes into disjoint elements. The data plane and control plane are divorced, and the decision making power that signifies the control plane is moved to a higher level. This is the controller level (network operating system level), and is what allows SDN to have a global view of network traffic. Further, this separation allows the management plane to define network policies and instruct the control plane to enforce them remotely through a southbound interface. Remote action of this type requires a special control channel be created between the control plane and the data plane, and this channel is known as the southbound interface. Given the young stage at which SDN currently is, the control channel discussion is tightly bound to the control protocol used. This work is limited to the OpenFlow v1.3 protocol [14], so this channel is generally known as the OF channel. Once the forwarding devices have received their configurations from the control plane through the OF channel, they operate as L2/L3 devices, forwarding traffic according to the instructions received from the controller. This architecture is shown in Figure 2.1.

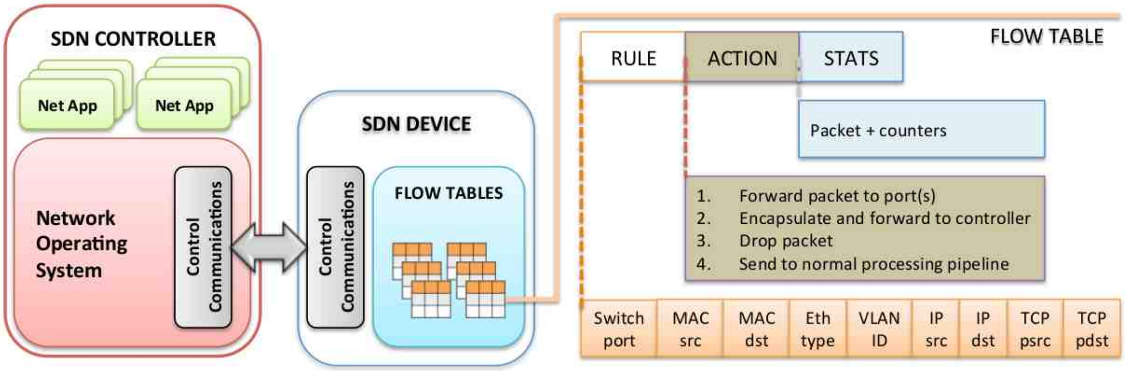

Fig. 2.2. OpenFlow-enabled SDN devices [13].

The OpenFlow v1.3 (OFv1.3) SDN protocol is one of the most accepted southbound interfaces and features a subnetwork of OFv1.3 switched network devices. The switches operate on a basic network entity called a flow, to whose belonging a packet must be determined. Each OFv1.3 switch contains one or more Flow Tables, whose entries are of the rule, action, statistics tuple seen in Figure 2.2. An incoming packet may belong to a flow, or it may not. As the incoming packet arrives, it is first applied to the lowest identity flow table. The match criteria of each flow entry is applied to the incoming packet, and if the incoming packet does belong to a flow, the switch takes the corresponding action(s) to process the packet. If the incoming packet does not match a flow in the lowest identity flow table, it may be sent to the controller in a PACKET_IN message or it may be dropped, depending on configuration. While the OFv1.3 switch awaits a response (PACKET_OUT), from the controller, it buffers the packet to minimize traffic disruption [14]. In this way, OFv1.3 switches operate as both L2 and L3 forwarding devices.

2.3 Cloud Computing

Cloud computing (the cloud) is a network-based [12] computing model that is characterized by many users sharing resource time on very large compute and network systems [15]. Most cloud users are enterprise class applications whose function must be wholly independent of the underlying infrastructure, and must further be logically separated from each other. This model is enabled by the principles of SDN, machine virtualization, and network virtualization. Virtual machines (VMs) have the function of physical machines without the limitations of the physical machine's hardware. There is a performance tradeoff, depending on the mechanism of virtualization implementation, but broadly the cloud would not be able to function without this concept. The virtual machines are connected to the virtual machine manager (VMM, also known as the hypervisor) through a virtual network, composed of virtual network devices which are kernel services whose software completely mimics the functionality of physical network devices. In this way, the VMM can create, destroy, maintain, and migrate virtual machines completely independently of each other through software means alone [15].

2.4 Artificial Immune System

An Artificial Immune System (AIS) is a computer system which is inspired by biological immune systems and features many of their properties [16]. Among these properties are the abilities to learn, adapt, and tolerate error. The most basic function an AIS serves is that of classification, as the primary goal of an AIS is to detect malicious antigens, which include pathogens, toxins, foreign substances, etc. Most generally, the classification is binary and seeks to classify the unit in question as “self” or “non-self” and is known as the self-nonself discrimination problem [17].

In biological systems, this classification decision is made by immune cells, and, once an entity has been classified as an antigen, the immune cells collaborate to destroy it. In this way, immune cells serve dual purposes: detection and intervention. Detection is done using a pattern matching process. In this pattern matching process, detector immune cells classify other cells as antigens if they affirm to match a given non-self pattern, rather than if they fail to match self patterns. The pattern matching process involves searching the cell’s surface for a contiguous set of specific chemical receptors. Because of this, the more non-self space the detector cell’s search pattern generation method can cover, the more effective the detectors will be at detecting antigens. The non-self search patterns are generated randomly, and undergo a training process called negative selection [16] to eliminate those that result in false positives (classifying self cells as non-self cells). If a detector cell successfully passes the training process, it is released into the immune system. This results in a system in which cells are in one of two states: training and released.

In an AIS, the tasks of detection and intervention are split, and the artificial immune cell’s primary job is distilled down to the job of detection, while the intervention task is left to another component of the system. As a result, these artificial cells are known as detector agents. Detector agents closely mimic the behavior of the biological system. They classify based on pattern matching, are randomly generated, undergo a process of negative selection training, and are only released into the production system once they have passed all test criteria. Originally, the pattern matching process also closely mimicked the biological system and employed a method of r-contiguous bit string matching [16], but more effective match processes have since been developed [18] [19] [20] [21] [22]. Upon successful detection of an antigen, the AIS can take a specific action. This could involve further classification of non-self cells beyond the initial binary classification, with specific actions being taken based on this subtype. The normal action process involves destroying the newly discovered antigen and disseminating the detector to other parts of the distributed system to prevent against future or distributed attacks. This process is called immunization and in this way, as threats develop and evolve, the AIS can continue to learn and improve its set of available detectors beyond the initial set of detectors.

2.5 Self-Organizing Map

A self-organizing map (SOM) is a type of artificial neural network (ANN) [23]. The central component of an ANN is the ability to learn and adapt based on incoming data. Among the learning process architecture categories are feedforward, feedback, and unsupervised. SOMs are of the third type: unsupervised. Here, neighboring cells compete and adjust based on the activities of their neighboring cells, and develop specific detector patterns as a result. These detector patterns provide a mapping from a high dimensionality input learning space to a low dimensionality output space in a process known as dimensionality reduction. Like the other types of ANNs, SOMs feature two operation states: training and mapping. These operational states correspond well with the cell states in the AIS.

2.6 Network Simulator 3

Network Simulator 3 (NS-3) is an open-source software network simulator developed and maintained by the NS-3 Consortium for the purposes of network research and education [24]. It provides realistic packet and traffic modeling, and is well suited to building, configuring, and testing networks that would otherwise be difficult or expensive to physically build and test. NS-3 operates as a discrete event simulator based on an event queue. This means network time (also known here as virtual time) proceeds forward in discrete steps, and proper maintenance of the event queue is essential to simulation event progress.

NS-3 takes great care to model physical networks as closely as possible. As a result, simulated network building follows the same basic steps in physical network building. Simulated network entities called nodes need network devices akin to physical network boxes requiring network interface cards. These network devices may operate on separate channels and feature realistic propagation delay and loss models, and mobility modeling, all with the highly capable monitoring mechanism that all software affords. Further, NS-3's network modeling is realistic enough to support external network entity integration. This affords the ability to test physical switches, routers, and hosts using NS-3's simulated networks though kernal TAP/TUN interfaces [25] popular with other virtual networking applications.

2.7 Related Work

This section explores related work in developing SDN network-based [12] or AIS intrusion detection systems (IDSs).

The work of Jha and Archaya [26] builds an AIS-based IDS featuring network-based [12] monitorization and unsupervised learning they call I3DS. I3DS is built around the function of two modeling techniques, one probabilistic-based and one decision tree-based. The probabilistic technique uses a Hidden Markov Model [27] structure to adaptively learn and provide a preliminary self vs non-self classification on novel traffic. The decision tree structure then takes the non-self candidate data and performs a final self non-self classification. I3DS is then compared against other learning-based IDS techniques (k-means, SOM, etc.) in testing utilizing the KDD Cup 99 Training Dataset [28] from [29]. I3DS outperforms all competition. Though

I3DS is purely theoretical, and thus does not offer a network monitoring scheme, it should be the goal of this work's SOM module to outperform I3DS in the same test conditions.

Hong [30] develops a network-based [12] malware detection method that detects anomalies in Domain Name System (DNS) traffic, rather than IP application traffic. The method (DTA) uses captured DNS traffic to build two graphs: DNS Lookup and DNS Failure. DNS Lookup is a weighted bipartite graph [31] whose edge set provides a mapping from a domain name to its corresponding IP address, and logs the query's frequency for a given time period. As DNS Lookup can only contain information on successful queries, DNS Failure is created to provide a mapping between failed DNS queries and the hosts originating the query, where the edge weight represents its frequency. Once created, the graphs are monitored for presentation of well-defined graph artifacts that are known to correspond with malicious network activity. However, graph presentation of these artifacts alone is insufficient to complete self non-self classification, so a decision tree module is included. While this method offers promise, it is a purely theoretical model and does not offer any kind of network monitoring scheme. DTA should ultimately be outperformed by the work of this project.

[32] builds an OpenFlow-based DDoS detection method utilizing the SOM technique. Their SOM is trained using a 6-tuple of native OF switch statistics gathered from the NOX [33] SDN controller. This 6-tuple is composed of the following features,

1. Average Packets per flow. Due to the general DDoS process of generating many flows from many sources, each individual flow tends to have a low average number of packets per flow.

2. Average Bytes per flow. As application DDoS attacks intend to overwhelm destination terminal equipment resources such as sockets, memory, and CPU time, DDoS attack packets tend to have lower than average packet byte sizes.

3. Average Duration per flow. This feature is used to decrease the rate of false-positive events.

4. Percentage of Pair-flows. Pair-flows are flows that compose the one-way traffic in a two-way conversation. Most benign application traffic takes place as pair-flows, while most attack traffic is composed of single-flows, i.e.- does not elicit a target response. A low percentage of pair-flows may indicate a DDoS attack in progress.

5. Growth of Single-flows. This feature is related to the percentage of pair-flows. It is a time rate of change feature that tracks the rate of change in the number of single flows. As DDoS attacks generally flood a destination with single-flows, a sharp rise in the rate of single-flows may indicate a DDoS attack in progress.

6. Growth of Different Ports. Since many DDoS attack packets are sent to random destination ports, a sharp rise in the number of destination ports used may indicate a DDoS attack in progress.

While effective, [32] does not allow for communication between multiple networks nor statistical methods which analyze OF switch ports to delineate attacking hosts from benign hosts. Further, this paper does not address the effects that DDoS attacks have on SDN controllers themselves.

SDN network architectures offer new tools with which to build network monitoring and anomaly detection mechanisms, but also cause a shift in the focus of DDoS attacks from the network forwarding devices to the controller. [34] addresses this issue by developing a quantity, entropy, which represents the number of PACKET_IN events per protected destination per time unit window. The object of a DDoS attack against an SDN controller is to overwhelm the OF channel bandwidth or controller processing capacity by generating an unserviceable number of PACKET_IN events. By quantifying the number of PACKET_IN events per host per unit time, attacks of this type can be monitored and detected. This project is effective in detecting attacks against SDN controllers and individual hosts, but it has limitations. This work cannot handle an attack against a whole subnet, has no communication mechanism for SDN networks with more than one controller, and does not have a mitigation strategy in the event of detection.

The work of [35] develops a DDoS blocking application that is not based on the detection of statistical anomalies as most traditional IDSs are. They have identified the difficulty in developing methods to produce accurate and precise statistical anomaly detection in the sphere of botnet-based attacks as the project's primary motivation. Instead, they show that a technique creating an SDN network application utilizing native SDN methods can block DDoS attacks when no statistical anomalies can be found. The network application properties follow:

- Generality. The technique uses only standard OF Application Programming Interfaces (APIs) and thus can be used in general OF-based SDN environments, independent of SDN controller.

- Semi-autonomous. The developed network application requires minimal destination server support in an effort to maximize server network and compute resource utilization.

- POX-based. The network application runs on top of the POX [36] SDN controller.

- Mininet tested. The network application has been tested using the Mininet [37] network emulator.

While lightweight and effective in their testing environment, this technique is inherently limited and not suitable for cloud environments. The main limitation is in requiring the network application to create a secure channel with the destination server to be protected. Not only does this violate SDN dataflow principles, it is neither practical nor desirable in cloud computing environments as the number of channels scales linearly with the number of destination servers, resulting in unacceptable overhead.

Saurabh and Verma [22] developed a proactive AIS-based anomaly detection and prevention system. A significant contribution of this system is its development of evolving, self-tuning detectors and the concept of detector power. The work further contributes a mechanism that allows detection agents to collaborate to detect and take action against statistical anomalies. Though they improve upon previous negative-selection algorithms [16] [19] [20] [21], the learning algorithm on which the AIS is based in this work is still a supervised learning algorithm. As such, its performance will always be based on the strength of the training set data it was initially taught with, and will adapt to new attacks more slowly than unsupervised learning techniques such as SOM.

3. CHALLENGES AND HYPOTHESIS

This chapter articulates the general challenges associated with designing a system capable of detecting DDoS attacks in cloud computing environments, and the specific challenges in developing and implementing the primary work of the design. It concludes with a statement of the hypothesis the project intends to affirm.

3.1 Primary Challenges

The primary challenges are those identified as the basic and essential tasks a general system must perform in order to protect a cloud network from a DDoS attack.

3.1.1 Monitorization

Sufficiently monitoring network behavior is the first challenge to be met. The network monitoring scheme is the ultimate source of all information from which AIS decisions are made, so no data down the data pipeline may be of a finer resolution than the monitoring scheme itself supports. Further, the more network traffic monitoring data that can be generated, the more applicable the resulting model represents the network. As cloud virtual networks become more complex and more distributed, no single network device has a global network view, and destination-based monitoring becomes infeasible. Instead, network-based [12] monitoring methods composed of multiple distributed monitoring agents must be implemented.

3.1.2 Detection

To effectively secure a system from attack, the security system must be able to accurately and precisely predict when an attack is going to take place, or detect an attack in progress before damage is done. Due to the imperfect nature of prediction and detection, an amount of error will invariably exist. There are four types of detection events [26] [30]:

1. True Positive (TP)– the system detected an event that occurred
2. False Positive (FP)– the system detected an event that did not occur
3. True Negative (TN)– the system did not detect an event that did not occur
4. False Negative (FN)– the system did not detect an event that occurred

The error types are related in a trade-off scheme. To achieve a higher TP rate, a system would likely experience an increase in FP rate as well. The situation is the same for FP and FN rates. Based on the type of network traffic, some detection applications will be more tolerant of certain error types than others. For example, a government agency internal network may have access to sensitive data. Such an environment would have a very low tolerance for FN type errors, with the understanding that an increase in FP type error tolerance would be necessary. Conversely, a public media server would seek to minimize FP errors, and would likely have a higher tolerance for FN errors. Due to this reality, the detection module should be configurable such that it can be tuned to the performance requirements of the specific application implementation environment.

3.1.3 Intervention

An effective system must be able to take action against an identified threat. The action taken must be able to stop or mitigate the harmful effects of the attack and should provide some sort of attack memory should a similar attack occur in the future. Given a distributed environment, the action taken should include a broadcast of the attack signature to potentially affected network devices such that other network devcies may stop or mitigate the attack. The attack mitigation technique should maximize effect on the targeted threat while minimizing the disruption to normal (benign) traffic.

3.2 Hypothesis

We aim to meet all core challenges by developing an SOM modified AIS OpenFlow network application with a monitorization module utilizing native SDN features in OpenFlow v1.3 to both monitor and intervene. This work hypothesizes that an SDN environment with OFv1.3 compliant switches is capable of monitoring a cloud environment's network to a level sufficient to enable an SOM based AIS network application to detect a DDoS attack in progress. The AIS module resides on top of the OF network controller as a network application. Upon successful detection, the AIS network application defines a new network policy capable of mitigating the attack and then instructs the OF controller to enforce the attack mitigation policy. Native OFv1.3 switches are sufficient to execute the monitorization scheme and mitigation policy without custom development.

4. PROPOSAL

This work seeks to support its hypothesis by building a proof-of-concept system. This system is intended to demonstrate the effectiveness of an OpenFlow v1.3 based network application in monitoring network traffic and detecting DDoS attacks in progress, assuming a suitably accurate detection signature is provided by the SOM module.

4.1 Collaborative Profiling

This project seeks to build its proof-of-concept system using a network-based [12] detection scheme within an OpenFlow v1.3 switched network. In contrast to traditional ingress switch or destination-based methods, network-based [12] methods compile traffic data from many distributed traffic monitors. To take advantage of OpenFlow's strength in centrally handling distributed network architectures, it proposes the monitorization scheme be based on OpenFlow flow statistics collection native to the protocol.

The proposed framework develops a collaborative traffic monitoring model for attack detection, shown in figure 4.1. In it, each OF switch features a profile extractor that collects flow statistics for a subset of the network traffic flowing through the switch. The profile extractor is implemented as a low table ID, high priority flow table entry, and does not perturb any network traffic flowing through the switch. The OF switch collects the desired traffic data according to native OpenFlow matching rules on fields given in Table 4.3 on page 32. Periodically, a profile aggregation network application instructs the SDN controller to collect the captured data from each switch, then aggregates the data to obtain a global network view of network traffic. In this way, network-based [12] monitorization is achieved and the monitorization primary challenge is met.

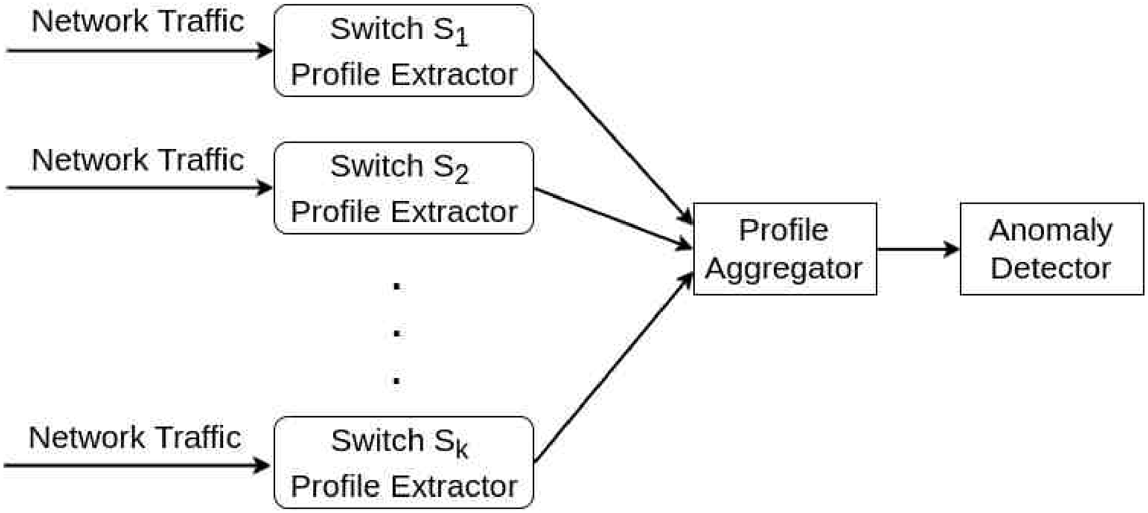

Fig. 4.1. Collaborative traffic analysis model.

4.2 Toxicity

The nature of DDoS attacks is to overwhelm a computing resource. If a sufficient amount of traffic arrives at the attack target within a defined period of time, the destination becomes overwhelmed and inaccessible. This concept of time is very important, as 10Mb of network traffic in ten seconds is a very different task to handle than 10Mb in one millisecond. Native OpenFlow flow statistics are able to give precise packet counts, byte counts, and even average bitrates. While these figures may be able to give a good instantaneous snapshot of network traffic, they alone are not enough to track the overwhelming effects of time sustained abuse. To track a flow's capacity to overwhelm, we define a function of time: toxicity.

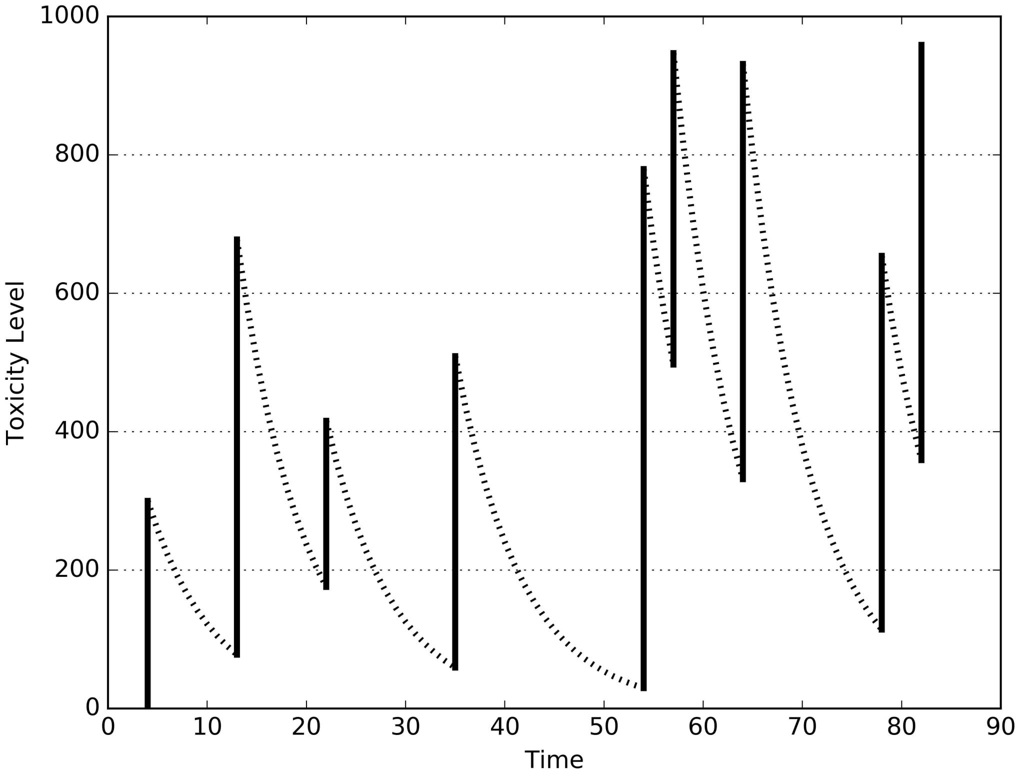

Fig. 4.2. Example of the toxicity level's linear increasing and exponential decreasing behavior.

A single packet traveling through a network requires a non-zero amount of resources in order to be processed by network devices including switches, wires, routers, and destination servers. This amount varies based on packet size, L2, L3, and L4 protocols, etc, and may be quite small per packet, but can accumulate to an overwhelming amount very quickly. It becomes very useful to think of a packet as containing a fundamental amount of inherent toxicity. As more packets travel through a network en route to a final destination server, the general toxicity of the network increases. As more requests arrive at a server than it can handle per unit time, the toxicity level accumulates at a rate proportional to the number and type of packets it is receiving.

If traffic increases enough, the destination server becomes overwhelmed and becomes unusable, an event analogous to poisoning. As network traffic subsides, the toxic effects subside as a function of time as the server is able to clear its queue. Even TCP SYN flooding fits the model by simply assigning a higher fundamental toxicity value to the TCP SYN request than a normal TCP, UDP, or ICMP packet. In seeking to protect a resource from attack, it becomes useful to think about the problem as trying to keep the resource's toxicity level from reaching a poisonous threshold. As general existing IP and SDN monitoring mechanisms are effective in keeping network infrastructure from being overwhelmed, our focus is on protecting the destination servers themselves.

Toxicity is modeled by considering how destination servers receive and process service requests. The amount of fundamental toxicity a flow accumulates with each packet reception is related to the server's maximum queue size, the packet size, and the L4 protocol, but not to the current size of its buffer or queue. Therefore, toxicity increases with each event. Further, toxicity levels are kept for both packet statistics and byte statistics, and thus we define two values, τ_P and tau_B for packet toxicity and byte toxicity, respectively, as,

$$\tau_P(t) = \tau_P(t - \Delta t) + b_P(t) \tag{4.1}$$

$$\tau_B(t) = \tau_B(t - \Delta t) + b_B(t) \tag{4.2}$$

where $\tau_P(t)$ is the packet toxicity level at time t, Δt is the elapsed time since the last toxicity update, and $b_P(t)$ is the fundamental packet toxicity deposit amount for this flow. The byte toxicity calculation follows the same logic as the packet toxicity, and the deposit equations are given below.

$$b_P(t) = \alpha \frac{Packets_t - Packets_{t-\Delta t}}{\Delta t} \tag{4.3}$$

$$b_B(t) = \beta \frac{Bytes_t - Bytes_{t-\Delta t}}{\Delta t} \tag{4.4}$$

where α and β are configurable constants.

Toxicity dissipates at a rate relative to the number of packets in the queue left to be processed and the time to service one request. Since the rate is relative to its current value, this process is modeled by the differential equation,

$$\frac{d\tau}{dt} = -\lambda\tau \tag{4.5}$$

with solution,

$$\tau(t) = \tau_0 e^{-\lambda t} \tag{4.6}$$

where λ is the exponential decay constant. Combining increasing and decreasing factors gives the composite function for a flow's toxicity as a function of time,

$$\tau_P(t) = \tau_P(t_0)e^{-\lambda(t-t_0)} + b_P(t) \tag{4.7}$$

$$\tau_B(t) = \tau_B(t_0)e^{-\lambda(t-t_0)} + b_B(t) \tag{4.8}$$

where t_0 is the time of the most recent update. In this way, an accurate, configurable model capable of capturing the ability of a flow to overwhelm a destination server is developed. An example of the toxicity level's time series behavior is given in figure 4.2.

4.3 System Architecture

The proposed system architecture is composed of an OF v1.3 SDN NS-3 network, a Floodlight SDN controller, and an AIS network application module. The OF v1.3 network simulates the behavior of a UDP flood DDoS attack on a single target, and includes normal background UDP traffic. The NS-3 network features ofswitch13 [38] OF v1.3 switches that are externally controlled by the Floodlight SDN controller [39] running in the same Linux user space as the NS-3 simulation. The NS-3 Floodlight interface is made through a Linux TAP interface [25], whose L2 and L3 addresses NS-3 associates with the controller through its native ns3::TapBridge module. The AIS network application module interfaces with the Floodlight controller via HyperText

Transfer Protocol (HTTP) REpresentation State Transfer (REST) API to monitor the network and detect DDoS attacks in progress. Figure 4.3 shows the system block diagram.

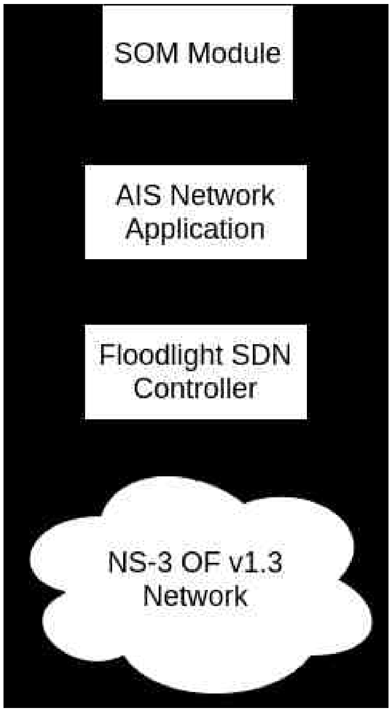

Fig. 4.3. System Architecture

The Ais process is given in figure 4.4. The process is split into two logical entities, the central coordinator is composed of the network controller and network application (including SOM module), while the distributed detectors are the OF v1.3 switches. Potential detectors are initially generated by analysis of an initial dataset, and continually generated by analysis of network traffic. These potential detectors compose the Self Set. The Self Set undergoes the negative selection training process to become the Antibody Set, which is then distributed to the appropriate distributed detector(s). The detectors then collect flow statistics on incoming traffic, and match flow patterns against their antibody subsets. Flows that do not match antibody patterns are processed normally according to OpenFlow v1.3 processes, and flow statistics are periodically gathered and sent to the central coordinator for continued adaptive learning. Upon discovering a flow that does match an antibody, the detector alerts the central coordinator and the expressed antibody is added to the pathogen library.

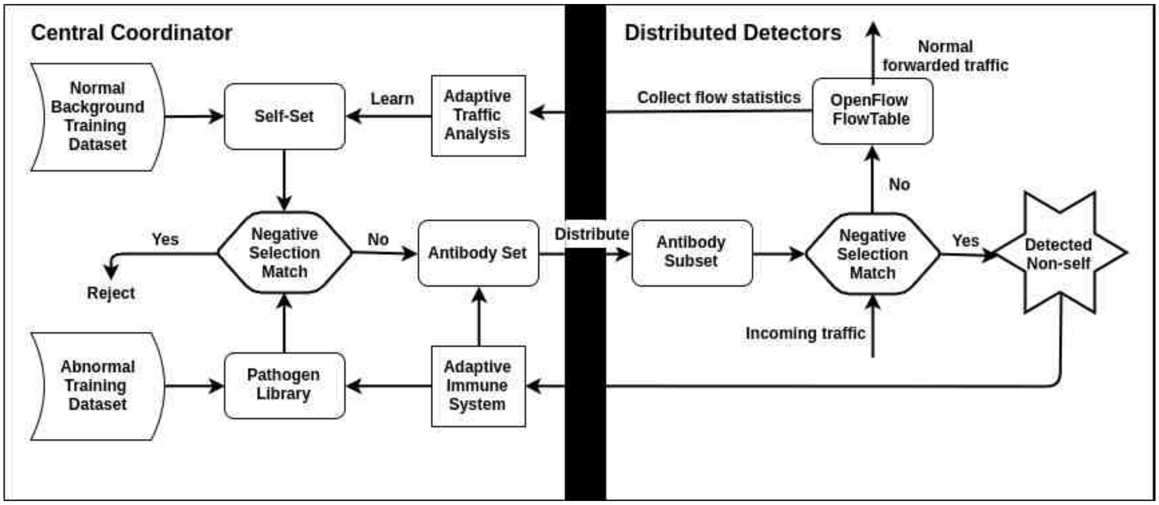

Fig. 4.4. Ais process.

4.3.1 Network Architecture

The NS-3 network built for this project simulates the behavior of a UDP flood type DDoS attack converging on a single target. The network features ofswitch13 [38] switches externally controlled by the Floodlight SDN controller. In keeping with general SDN practices, the data and control planes are logically separated into independent networks. This type of network in which control signaling is carried in a different network than user data is called an out-of-band network [40]. Out-of-band networks, while good general SDN practice, afford the additional benefit of control plane insulation against application-level DDoS attacks. An entire in-band control network could easily be taken down by an application-level DDoS attack, as the OF control messages would be dropped due to attack congestion. The control network, also called the OpenFlow channel, is configured as a single Carrier Sense Multiple Access

(CSMA) ethernet [41] channel. The OF switch L2 and L3 addresses are configured within the NS-3 simulation, while the controller L2 and l3 addresses are configured by the Linux TAP interface, external to the NS-3 environment. Any changes to the OpenFlow channel topology must be reflected in both the TAP interface and the NS-3 environment. The control plane network topology is given in Figure 4.5.

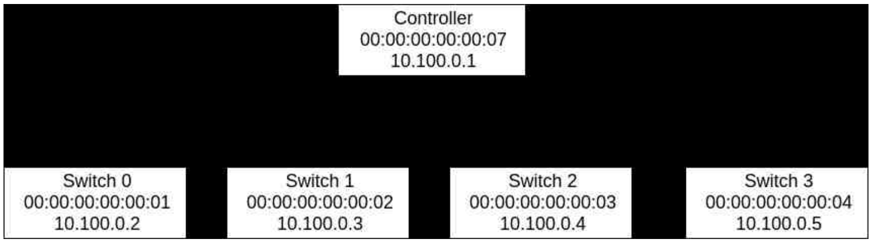

Fig. 4.5. Network Control Plane Topology

The data plane network topology is contained entirely within the NS-3 environment. In it, each host is connected to a single switch by a dedicated CSMA ethernet channel. The dedicated channel allows for high traffic monitor resolution, suitable for this proof of concept system. The host switches are connected to the target switch in a star topology, again with dedicated ethernet channels. The target switch then uses singly dedicated ethernet channels to connect with the destination servers. The network data plane topology is given in Figure 4.6.

4.3.2 AIS Network Application Data Structure

The AIS Network Application is intended to enable decisions to be made on large networks which generate encumberingly large volumes of network statistics (metadata). As the entirety of this metadata must travel over the limited OpenFlow channel, care must be taken to minimize the channel bandwidth utilization by the module, or risk interfering with normal network operations. Further, as network, memory, and CPU resources are marketable commodities in cloud computing environments, the application's data and processing considerations must also be taken into account. For

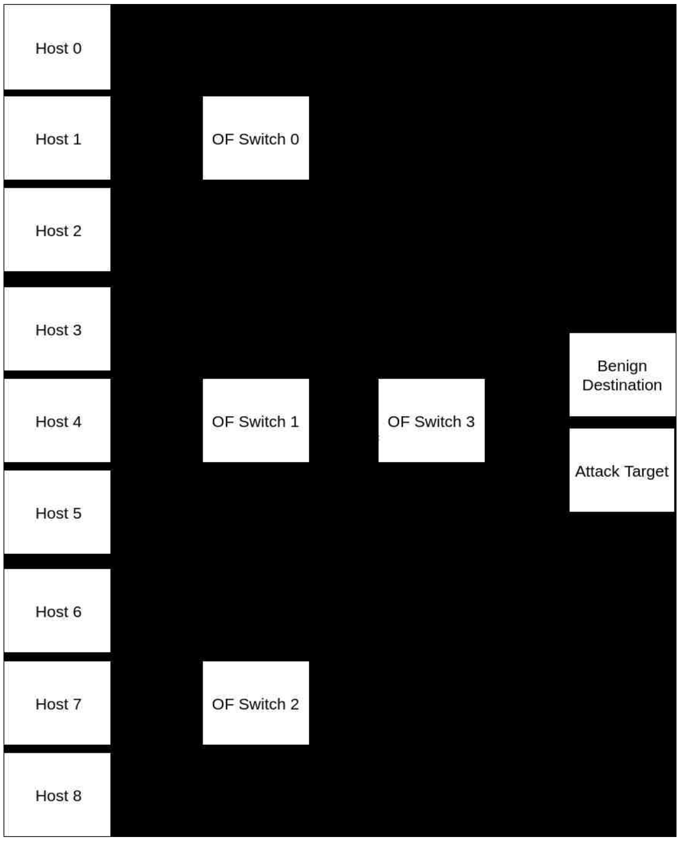

Fig. 4.6. Network Data Plane Topology

the end system, every effort should be made to reduce resource consumption wherever possible. As a result, the proof-of-concept AisTable and associated entries are developed with this in mind. The structure of the an entry into the AisTable is given in Table 4.1.

Table 4.1.
AisTableEntry fields

Signature	[SwitchCountsEntry]	Packet Toxicity	Byte Toxicity

The AisTable is a hash table [31], keyed on a hashing of the AisTableEntry::Signature. Each network profile (see section 4.1) of interest generates a unique Signature and thus a unique AisTableEntry in the AisTable. Table 4.2 gives the possible fields a Signature might contain. These fields are developed from the OXM match fields required by OpenFlow v1.3 specification [14], and represent the maximum view resolution the AIS system has into the network. A Signature instance may contain as many Signature fields as desired, but must contain at least one.

Table 4.2.
Signature fields (required OXM match fields).

Field	Description
OXM_OF_IN_PORT	Ingress port. Physical or switch-defined logical port.
OXM_OF_ETH_DST	Ethernet destination address.
OXM_OF_ETH_SRC	Ethernet source address.
OXM_OF_ETH_TYPE	Ethernet type of the OpenFlow packet payload.
OXM_OF_IP_PROTO	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	IPv4 source address.
OXM_OF_IPV4_DST	IPv4 destination address.
OXM_OF_IPV6_SRC	IPv6 source address.
OXM_OF_IPV6_DST	IPv6 destination address.
OXM_OF_TCP_SRC	TCP source port.
OXM_OF_TCP_DST	TCP destination port.
OXM_OF_UDP_SRC	UDP source port.
OXM_OF_UDP_DST	UDP destination port.

Due to the collaborative nature of the network-based [12] monitorization scheme, a single Signature may receive traffic data contributions from separate switches throughout the network. To allow for this, the AisTableEntry includes an array of SwitchCountsEntries, with a unique SwitchCountsEntry belonging to a single switch for the given profile. The SwitchCountsEntry keeps track of packet and byte counts for the individual flow, and the timestamp when these network statistics were last collected. These data are necessary and sufficient to calculate the toxicity contributions per time aggregation event as part of the calculation of equation 4.7. The structure of the SwitchCountsEntry is given in Table 4.3.

Table 4.3.
SwitchCountsEntry fields

SwitchID	Timestamp	Packet Count	Byte Count

4.3.3 AIS Network Application Architecture

The AIS network application is built as a Python [42] module on top of the Floodlight SDN controller. The controller - AIS module communication is done via HTTP REST API calls over a Linux TAP interface. The software architecture is written to be modular, easy to read, and easy to maintain and extend. The module is written according to good model-view-controller (MVC) [43] software architectural pattern practices to ensure the user cannot manipulate AIS data directly, but can only check status information and trigger module communication events. The AIS network application architecture is given in Figure 4.7.

At regular intervals, the AIS module's FlowEntryReceiver polls the controller for network flow statistics. Upon receiving the request for flow statistics, the controller sends an OF read-state TCP message to each of its NS-3 OF v1.3 switches through the dedicated OpenFlow channel. The switches reply with the accumulated statistics kept for each entry in their flow tables, which the controller then forwards back to the FlowEntryReceiver as HTTP GET JavaScript Object Notation (JSON) [44] data. These JSON switch flow statistics are then passed to the Ais module through the Ais::Communicator::FlowEntryReceiver. The JSON data is then parsed and filtered

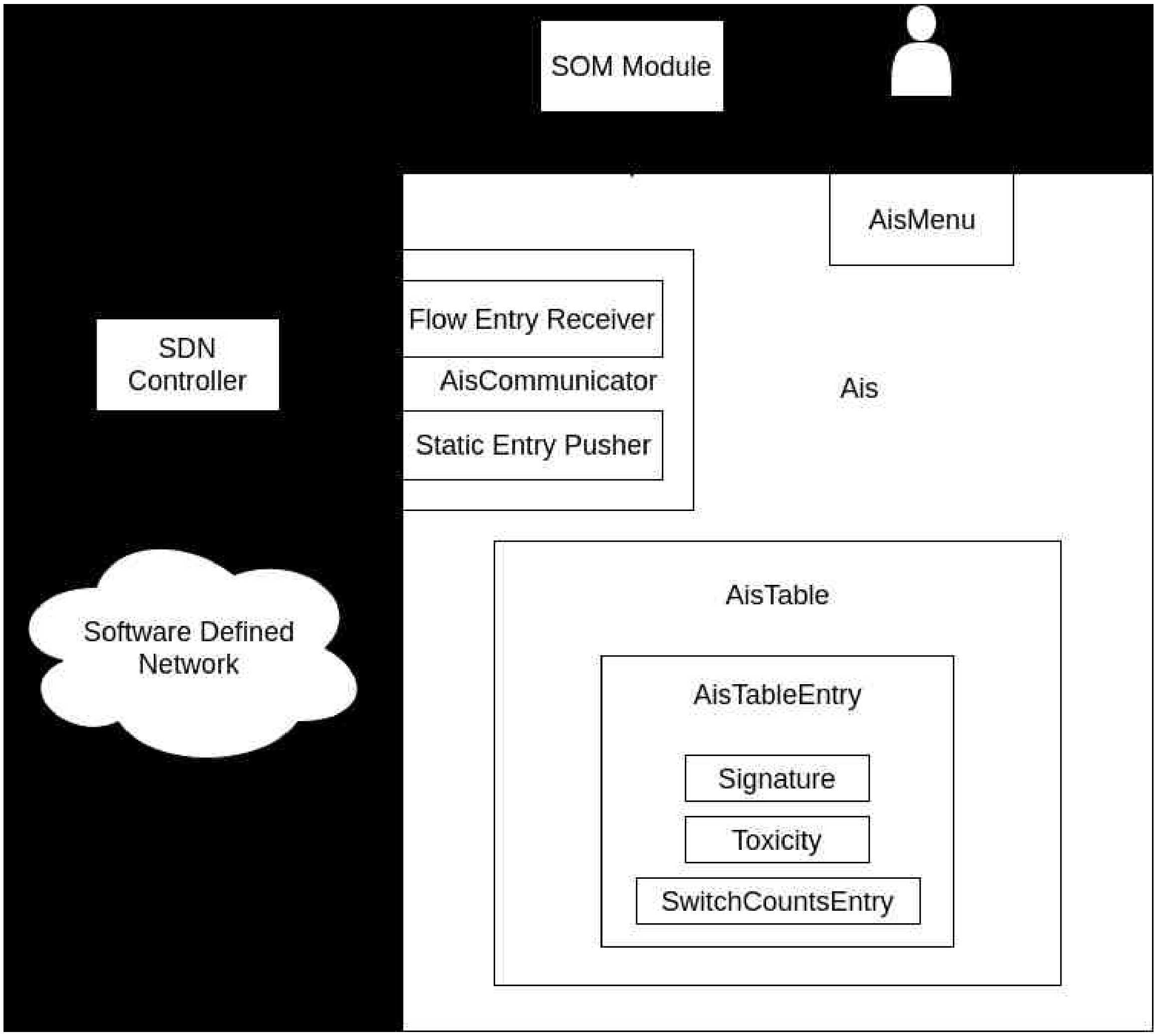

Fig. 4.7. Application Architecture

according to profile signature data stored within the Ais module. This allows only the profile statistics to be sent to the AisTable for toxicity updates. As part of the toxicity update process, the toxicity values are checked against the profile's toxicity thresholds. If toxicity thresholds have been surmounted, the AisTableEntry generates an appropriate hazard message. In accordance with project goals, the module takes no further action, though it does have the ability to define a new network policy which drops packets associated with the attack-classified profile. Implementation of the SOM detector generator module is not within the purview of this project, though a skeleton Python class is created and wired to the Ais module to ease future development.

5. EXPERIMENT

This chapter describes the experimentation designed to support the project's hypothesis. Included are descriptions of the experiment's topology, and specific limiting implementation challenges. Further, experiment results and analysis are presented that demonstrate support of the hypothesis.

5.1 Description

The project experimentation is designed to demonstrate the AIS application's nature as a network-based [12] AIS and its capabilities of meeting the monitorization and detection primary tasks. To accomplish these, experimentation is built on the integration of several software modules within a single Linux userspace, as described in chapter 4. The experimentation consists of a series of separate 900 second network traffic simulations in which network configuration and traffic patterns are generated by the NS-3 network as input, and the AIS module toxicity and hazard message responses are recorded as output. The project hypothesis is supported if:

1. *Monitorization*: the AIS is able to aggregate the distributed profile information into a network-based [12] global view of traffic.

2. *Detection*: the AIS is able to outperform destination-based detection methods for a sufficiently realistic simulated DDoS attack in the measured performance indicator metrics.

The NS-3 component gives the ability to simulate representatively realistic network traffic in order to test network configurations, external SDN controllers, and SDN network applications. To test the monitorization and detection capabilities of the AIS network application, the network of figures 4.5 and 4.6 is created. Two experimental traffic scenarios are then run.

5.1.1 Baseline Gathering Experiment Description

The first experiment develops a baseline for DDoS detection, and delineates the differences between network-based [12] and destination-based detection method performance. In this scenario, even numbered IP addressed hosts send a normal amount of UDP application traffic to the attack target and are to be classified as benign. Odd numbered IP addressed hosts begin by also sending a normal amount of UDP application traffic, but ramp up the amount of individual traffic as the simulation progresses. The input traffic patterns can be seen in figure 5.5. The simulation is run four times, with different attack types at the network level and different network monitoring techniques at the network application level. Scenarios tested are DoS attack with destination-based monitoring, DDoS attack with destination-based monitoring, DoS attack with network-based [12] monitoring and DDoS attack with destination-based monitoring. Results are given in section 5.3.1.

5.1.2 Profile Gathering Experiment Description

The second experiment feeds a more realistic traffic pattern to the system. In this experiment, a second destination is created, and is not the subject of an attack. Normal application UDP traffic flows from even numbered IP hosts to both the benign destination server and to the attack target. This traffic serves as background noise for the DDoS detection module, and should be neither detected as attack traffic nor disrupted as part of any attack mitigation efforts. This normal application traffic starts shortly after the simulation begins, and steadily continues through the simulation.

Attack application UDP flood traffic flows from odd numbered IP hosts to the attack target. As all host clients generate the same amount of individual network traffic, no single IP source address can be categorized as malicious. These flows come online in regular intervals, and reach warning and alert level thresholds at well defined times. It is the end goal of the project to be able to classify subflows within the detected flows, and filter out specific IP sources as a result. It is this ability that the SOM module is intended to provide, and thus is not required here.

However, a demonstration of the subflow classification behavior is possible before the SOM module is developed. This behavior is available through the concept of bit-mapping. OpenFlow v1.3 flow tables support arbitrarily bit-masked IP address match fields [14]. This allows flow table entries to match IP source addresses to specific bit patterns, rather than specific or wildcarded full addresses. These bit patterns can be generated randomly, and can go through a negative-selection training process in a "pseudo-SOM" module, with the bitmasks taking the role of detector. In following along the SOM AIS process, the selected bitmasks can be given to the AIS module. Once there, the AIS module can instruct the SDN controller to monitor IP addresses matching the bit pattern, rather than monitoring specific IP addresses or subnetworks. In this experiment, the pseudo-SOM module has generated such a bitmask, and the results of its use are given in section 5.3.2.

5.2 Implementation Challenges

In designing the experiments to support the project hypothesis, NS-3's inherent limitations as a discrete time event driven simulator place restrictions on real-world modeling. These limitations include the service request processing problem, discussed in 5.2.1, and the discrete time leap problem, discussed in section 5.2.2. Since all network events are software processed, large, highly parallelized networks cannot be simulated in real time, thus limiting network size. Further, the ofswitch13 module is young and not well matured. This module has placed additional constraints on network experimentation, most importantly the L3 subnetwork problem, discussed in section 5.2.3.

5.2.1 NS-3 Request Processing Problem

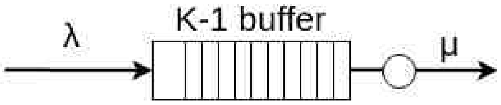

Fig. 5.1. Queueing model for a single server and its network packet buffer.

The service request processing problem concerns the ability of NS-3 to model the effects of a DDoS attack on network destination servers, which requires a discussion of queue modeling [45]. In physical destination servers, service requests arrive into the system as a random process *A(t)*, with interarrival times $\tau_1, \tau_2, ..., \tau_n$. It is sufficient for this experiment to assume the interarrival times are deterministic. These interarrival times give a system arrival rate of,

$$\lambda = \frac{1}{E[\tau]} requests/second \tag{5.1}$$

Once in the queue, the requests can then be processed by the server. The time it takes to process a single request (in our case, a single UDP datagram) is the service time, and is given by X. The rate at which the server can process service requests is finite and is defined as the processing capacity μ, where,

$$\mu = \frac{1}{E[X]} requests/second \tag{5.2}$$

Here, again, it is sufficient for this experiment to assume a deterministic processing capacity. The difference between the arrival rate and the processing capacity is the rate at which the queue's k-maximum capacity is filled. Once the queue's length reaches its maximum capacity, k, the system can no longer handle incoming service requests. Subsequent requests are then dropped according to the blocking ratio,

$$B = \frac{\lambda}{\mu} \tag{5.3}$$

It is this blocking ratio that fundamentally enables a DDoS attack. The target system is bombarded with requests until the arrival rate $\lambda >> \mu$, and the queue quickly reaches its capacity. At that point, the probability that any given request is blocked and therefore dropped is near 1, and the system is seen as unreachable.

In order to effectively model a DDoS attack, reception queue modeling must be implemented. A simple system, as in figure 5.1 is sufficient, though appropriate values of λ, μ and k must be determined. Further, the simulation environment must contain a mechanism of implementation. Natively, ns3::CsmaNetDevices undertake no such reception queue rate modeling, and do not contain a reception queue of any type [46]. As a result, incoming requests are serviced by the simulated application server as quickly as the simulator's hardware allows, independent of network virtual time. This leads to a restrictively high μ value. Thus, in order to simulate a DDoS attack, packets must arrive into the simulated target system at a greater rate than the simulation hardware's CPU can processs the packets. Recall that NS-3 is an event driven system, and that each event must be processed in series. Thus, in order for a DDoS attack to be modeled, the simulator must generate, send, receive, and process network traffic faster than its own hardware. It is therefore impossible for the effects of a DDoS attack on a server to be simulated without further NS-3 development to support the concept of reception queueing.

Development of appropriate reception queue modeling involves the addition of a reception queue to the ns3::CsmaNetDevice class. This reception queue contains configurable parameters for service processing capacity μ and buffer length k. As a result, this project experimentation defines an appropriate μ such that attack-level traffic causes the malicious effects of a DDoS attack on the target server, while allowing the target server to process network traffic under this level. Thus, in order to support the project's hypothesis, the AIS module must be able to detect when this level of network traffic is reached, preferably before the target server is actually rendered unreachable.

5.2.2 NS-3 Wall Clock Implementation

NS-3 is a discrete event simulator processing events in order from an event queue. Its virtual clock (simulated network time) progresses as a series of discrete steps, taking the value of the next event in the queue. Care has been taken to intelligently add events to the event queue such that network time is never allowed to advance backward, as in the physical world. However, due to its discrete nature, time leaps are possible in the simulator which much be avoided. For example, if the network clock is at time 10.0, and the next event is scheduled for time 18.4, the network clock will simply be advanced to 18.4 once the event at time 10.0 is finished processing. This is done without waiting the 8.4 second difference. This behavior must be avoided if the NS-3 simulation is to interact with an external entity like the Floodlight controller.

NS-3 does support a realtime operating mode in which these discrete time steps do not occur. This realtime module, however, contains inherent weaknesses that must be overcome in order for the project to progress. First, the network clock always begins at time 0. While this is sufficient to keep time, model events, and even handle some external communication, it is not suited to situations in which a timestamp must be passed in any sort of message. As this work may require current network time to be included in controller and/or AIS messages, and the module is not being actively developed, the work of this project must address this issue.

Time systems based on the UNIX Epoch [47] feature a counter-based clock concept which counts the number of seconds (or nanoseconds) elapsed since January 1, 1970. Linux systems concept of time revolves around the UNIX Epoch, and it is this clock that represents the real-world or realtime clock in this system. The NS-3 virtual (network) clock features a similar counter-based concept. It is an unsigned 64 bit

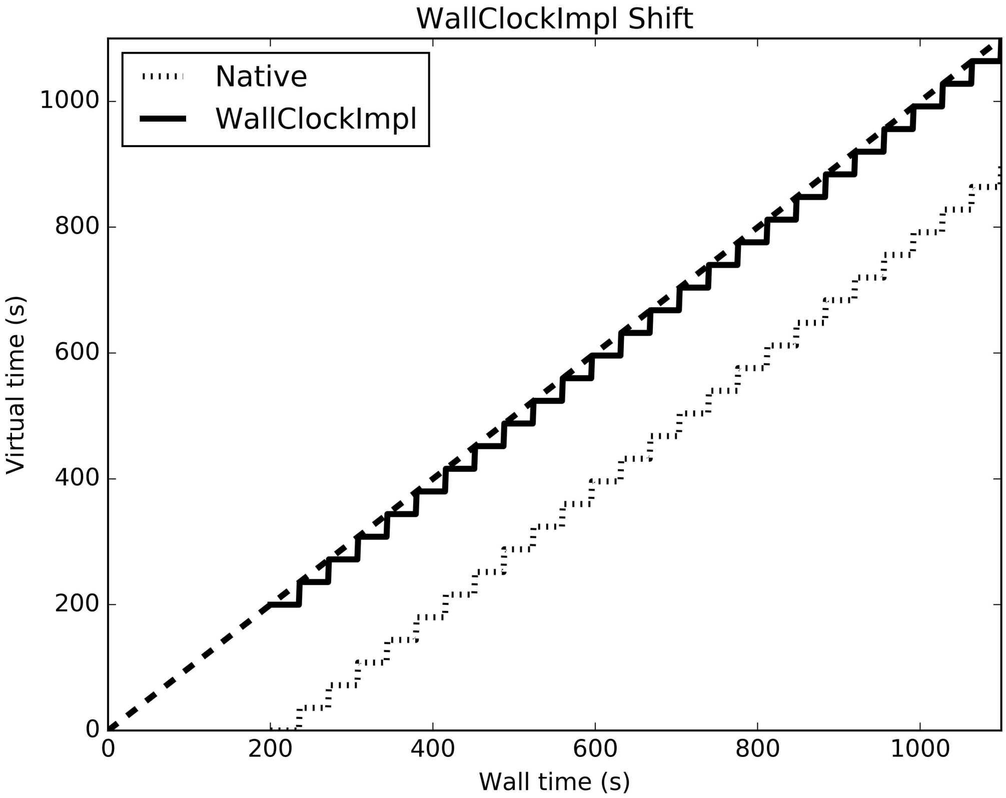

Fig. 5.2. NS-3 Wall Clock Problem seeks to shift the virtual clock up to a level such that the virtual and real clocks are equal.

integer representing the number of nanoseconds elapsed since the simulation began. This sets a network time resolution down to the nanosecond level, which is sufficient for this project. However, due to Epoch concept, it also entails that any external entity receiving timestamps from the simulation will think the timestamp came from 1970. Further, the native realtime module is catastrophically inaccurate. In testing with the SDN DDoS network built for this project, the realtime module lost approximately 9 seconds in one hour. This accuracy is insufficient for a network simulation that is intended to run multiple days or weeks.

The NS-3 realtime synchronization module was written inherently expecting the virtual clock to begin at 0. As a result, a significant portion of the production module had to be rewritten, and considerable research and developmental effort went into successfully solving the wall clock problem. Now complete, the new module is accurate to within 200 milliseconds per hour: a vast improvement and an accuracy that is within acceptable tolerance for this project. A visualization of the concept is given in figure 5.2.

5.2.3 NS-3 OpenFlow v1.3 Support (L3 Subnet Problem)

NS-3 has support for two OpenFlow switch specifications, a native OFv1.1 implementation [24], and an external module, ofswitch13 [38] which supports OFv1.3. Project study reveals that the native implementation is not well developed, and that the ofswitch13 module is superior. Though stronger than the native implementation, the ofswitch13 module is immature, and is under continued development.

The limiting ofswitch13 factor concerns the implementation of the ns3::CsmaNetDevice object. The CsmaNetDevice is the NS-3 equivalent of a physical ethernet Network Interface Card (NIC). In physical networks, ethernet NICs allow network elements to communicate via IEEE 802.3 [41]. Inherent in this communication is the ability to listen for channel activity and send, receive, and propagate L2 traffic. Further, these abilities are independent of inclusion in OpenFlow networks. A terminal network host is oblivious to the question of whether or not it resides within an OpenFlow network.

However, ofswitch13 implementation details do not partition the ns3::CsmaNetDevices in this way. Instead, host CsmaNetDevices, which should be oblivious to the OpenFlow details, are affected by the installation of the ofswitch13 module. This affect is not intended by the ofswitch13 developers, and results in a loss of the ability to send, receive, and propagate L2 traffic in the normal way. The effect is felt in network activity as an inability for host terminal equipment to send IP packets to a destination not within the sender's subnetwork. Upon investigation, it is determined that the problematic behavior is due to the module's poor design and speedy development cycle. Specifically, the ofswitch13's CsmaNetDevice redirects all packets to be processed by the OpenFlow logic, overwriting the necessary callbacks that allow the packet to be processed by the IP stack. As a result, the experimental network for this project cannot contain multiple IP subnetworks.

5.3 Results

Results are now given which lend support to the hypothesis that a network-based [12] AIS is capable of detecting and mitigating DDoS attacks in cloud computing environments. These results center around the detection performance indicators detection rate (DR) [26] [30], accuracy (A), precision (P) [26] [30], and F-score (F) [26] [30]. Where,

$$DR = \frac{TP}{TP + FN} \tag{5.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.6}$$

$$F - score = 2 * \frac{DR * P}{DR + P} = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.7}$$

DR represents the proportion of malicious packets the system is able to detect to all of the malicious packets in the system. Precision captures the likelihood that a packet detected as malicious truly is malicious. Accuracy represents the ability of the system to properly classify a packet as benign or malicious, and the F-score is a composite score that is used as a single "goodness metric". These performance metrics are composed of the four detection event types. Recall,

1. True Positive (TP)– the system detected an event that occurred

2. False Positive (FP)– the system detected an event that did not occur

3. True Negative (TN)– the system did not detect an event that did not occur

4. False Negative (FN)– the system did not detect an event that occurred

Central to the functioning of the AIS system is the correct implementation of the toxicity concept of section 4.2. Figure 5.3 gives the detailed behavior of the implemented toxicity level's behavior in unit testing. The figure displays the project's Ais::Toxicity class's behavior when the deposit amounts *b(t)* are independent uniform random variables (top) and deterministic (bottom). Both tests feature a deposit interarrival time that is deterministic, i.e.- independent random variables with a constant distribution.

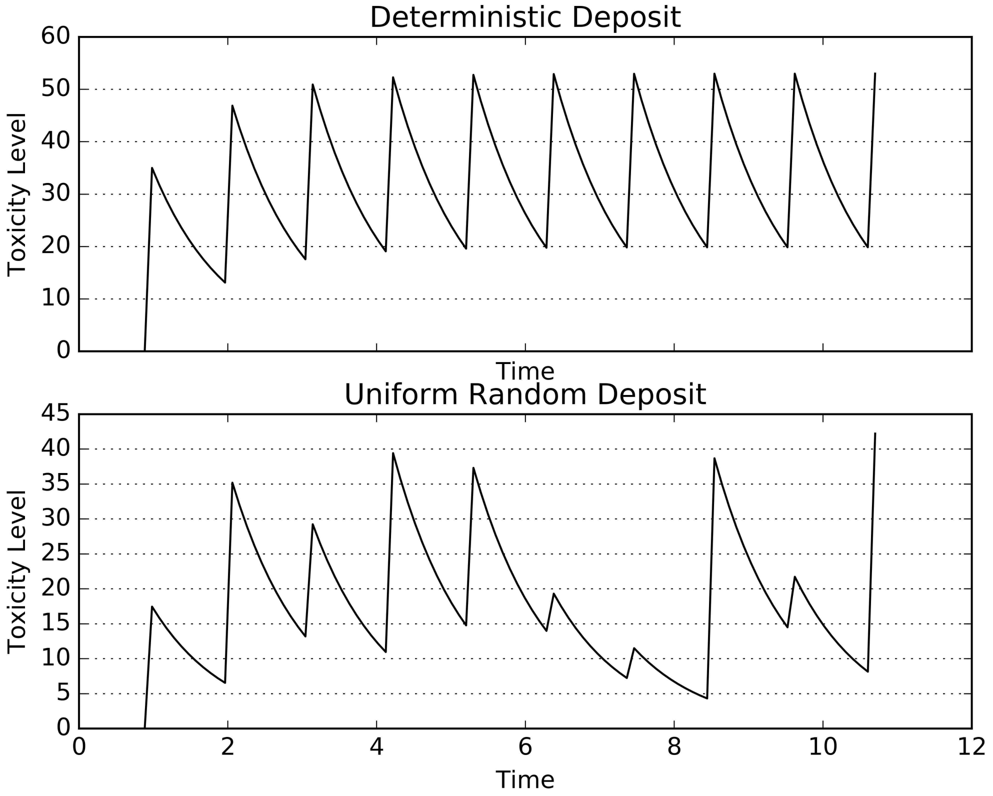

Fig. 5.3. Results of unit testing of the Ais::Toxicity Python class.

5.3.1 Baseline Gathering Experiment

The baseline gathering experiment feeds normal, attack pattern 1 (pattern 1), and attack pattern 2 (pattern 2) input traffic to the dual monitorization techniques of destination-based and network-based [12] monitoring. The AIS network application is fully configurable and provides both methods for testing purposes. Input traffic patterns can be seen in figure 5.4.

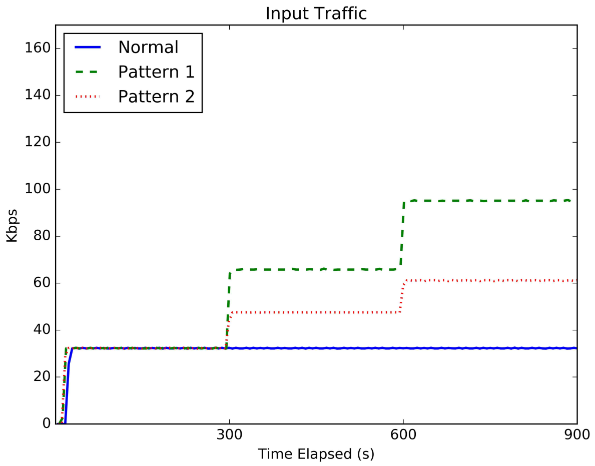

Fig. 5.4. Baseline gathering experiment destination channel traffic patterns.

In the pattern 1 simulation, a combination of normal and pattern 1 traffic patterns are given, as described in section 5.1.1. As can be seen in figure 5.5, the pattern 1 toxicity values follow pattern 1 input traffic in both pattern and magnitude, though a detection time gap of 6.2 seconds does occur. Similarly, the pattern 2 traffic detection patterns closely follow the pattern 2 input traffic pattern. Notice the pattern 2 toxicity values are lower than the pattern 1 traffic values for the same period. This is expected as the destination-based method does not aggregate the traffic, but only examines a single flow.

Similarly, the pattern 1 traffic detection matches the output of the destinatin-based method. These results are expected, as destination-based monoitoring methods are perfectly capable of detecting single source anomalous traffic. The difference is in

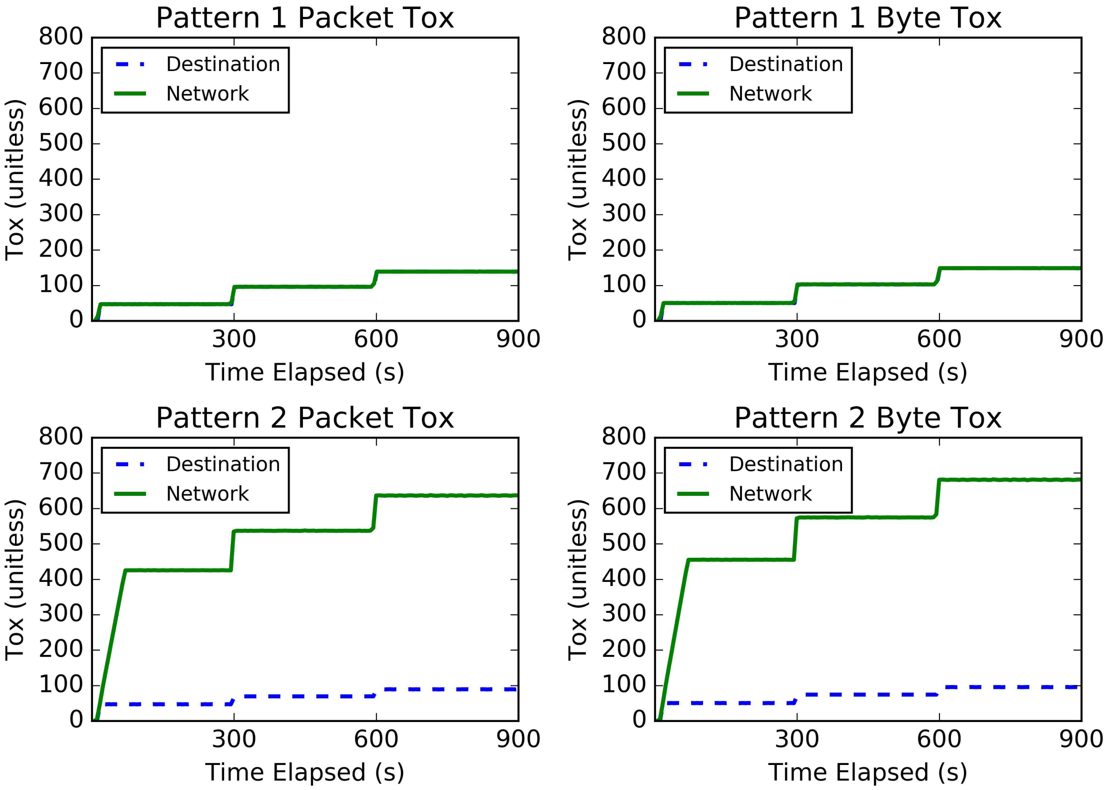

Fig. 5.5. Toxicity level results of baseline experiment.

the DDoS traffic detection. Though the overall network-based [12] and destination-based patterns match closely with each other, notice the large difference in magnitude. Since the network-based [12] method is able to aggregate a global view of network traffic, it is able to detect that the attack target is receiving a very high amount of network traffic, even while each individual flow may be below the single flow alert threshold.

Pattern 1 performance indicators for this experiment are given in table 5.1, which compares the two monitorization methods directly. Here, it should be noticed that overall performance is approximately identical for the two monitoring methods. The 336 packet difference in TP events is well within the 1389 event margin of error, which

Table 5.1.
Pattern 1 Performance

Base	TP	TN	FP	FN	DR	P	A	F
Destination	16698	17230	0	363	97.897	100.0	98.948	98.925
Network	17034	16951	0	458	97.382	100.0	98.670	98.673

is obtained by considering a 5.0 second AIS poll time and 3.6 millisecond interarrival time. Similarly, the TN and FN are within this margin of error. As a result, network-based [12] monitoring methods cannot be said to offer superior performance for this type of attack.

Table 5.2.
Pattern 2 Performance

Base	TP	TN	FP	FN	DR	P	A	F
Destination	0	130850	0	54712	0	0	70.516	0
Network	53648	109557	22689	270	99.500	70.278	87.670	82.374

Pattern 2 performance indicators, by contrast, do illustrate the performance gains offered by network-based [12] methods, and are given in table 5.2. Here it is seen that the destination-based method is unable to detect any malicious activity, thus both TP and FP event counters are zero. This gives a DR of 0 for this method, which is insufficient for the needs of cloud computing environments. Its accuracy of over 70% is deceptively high, as the majority of packets sent in this experiment are benign and thus should be counted in the “negative” event category. In examining the network-based [12] results, it should be noticed that the DR is above 99% and the accuracy above 87%. Though these figures are encouraging, it must be remembered that the AIS is configured for a low FN tolerance. This configuration results in the significant number of FP events seen, and the resulting precision is an ineffective 70%. Future development of the SOM module should reduce these FN events, and raise both the accuracy and precision of the system. Nevertheless, the network-based [12] method thoroughrly outperforms the destination-based method, and the hypothesis is supported by this experiment. Moving forward, destination-based monitoring methods are not considered.

5.3.2 Profile Gathering Experiment

The profile gathering experiment features the same network-based [12] AIS, and largely the same network configurations as the baseline gathering experiment. However, it differs in that it includes a benign destination server. This new server receives a steady amount of benign UDP application traffic throughout the simulation from each of its hosts. The benign destination has the IP address 10.1.1.11, while the attack target has the IP address 10.1.1.10. This experiment examines the performance of two detection profiles on this new traffic scenario.

The first, WILDCARD, offers the simple network-based [12] detection featured in the previous experiment. Its IP source address field is wildcarded, resulting in this profile being unable to classify subflows or monitor specific IP source addresses. The second, BITMASK, offers the pseudo-SOM bitmask concept discussed in section 5.1.2, and is expected to outperform WILDCARD. The AIS is seeded with these detection profiles in turn and identical network simulations are run on each.

Figure 5.6 gives the details of the WILDCARD profile experiment. Total network traffic is shown, as is a breakdown of traffic by destination, at the furthest resolution offered by the profile. Each plot in the figure features ALERT and down lines, which illustrate the time at which the AIS detected an incipient attack and the time at which the target server was taken down by the attack. The accumulated total number of positive detection events is also shown. Packet and toxicity curves are given next, with the general patterns closely following the input traffic patterns.

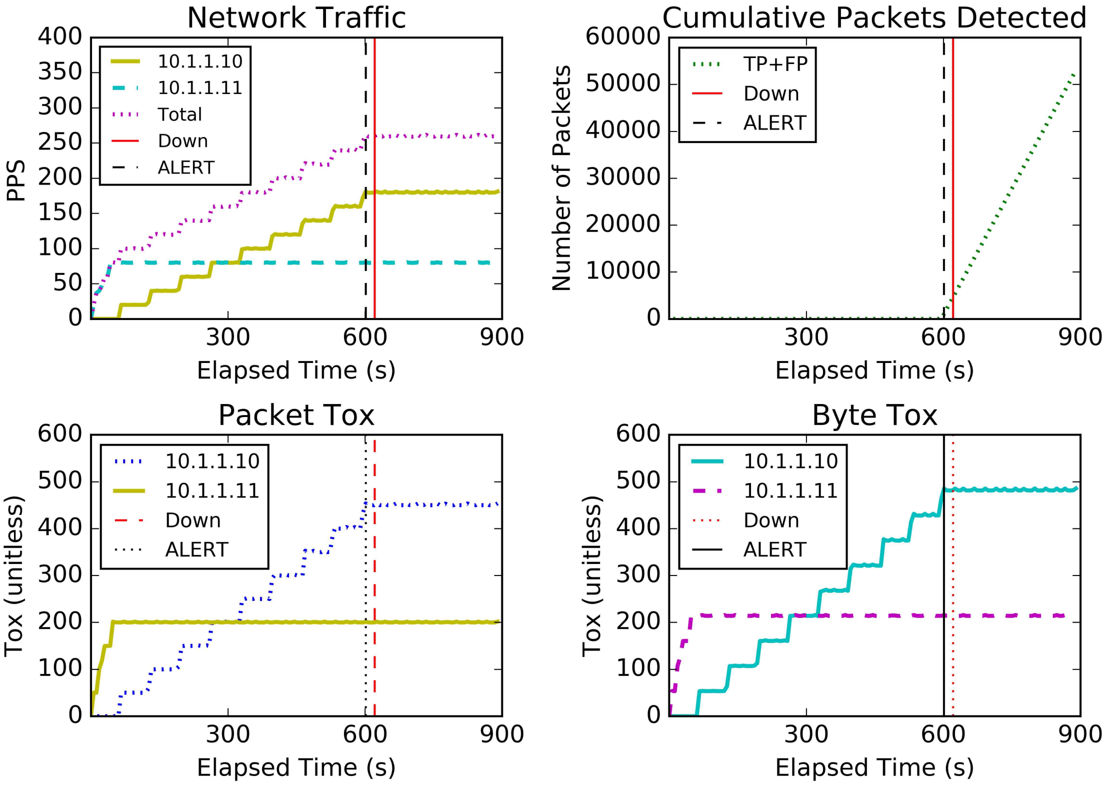

Fig. 5.6. WILDCARD Single profile with no profile bitmasking.

The results of the BITMASK profile are given in figures 5.7 and 5.8. It runs an identical NS-3 network configuration, so the network input traffic is the same. In the same method as the previous, this traffic is broken down by destination at the furthest resolution offered by the profile. It should be noticed that BITMASK is able to distinguish subflows within the 10.1.1.10 flow: 10.1.1.10 A (attack) and 10.1.1.10 B (benign). The accumulated packet detections are given as well, and when compared against WILDCARD's result, show a more shallow linear slope as a result of its fewer detection events.

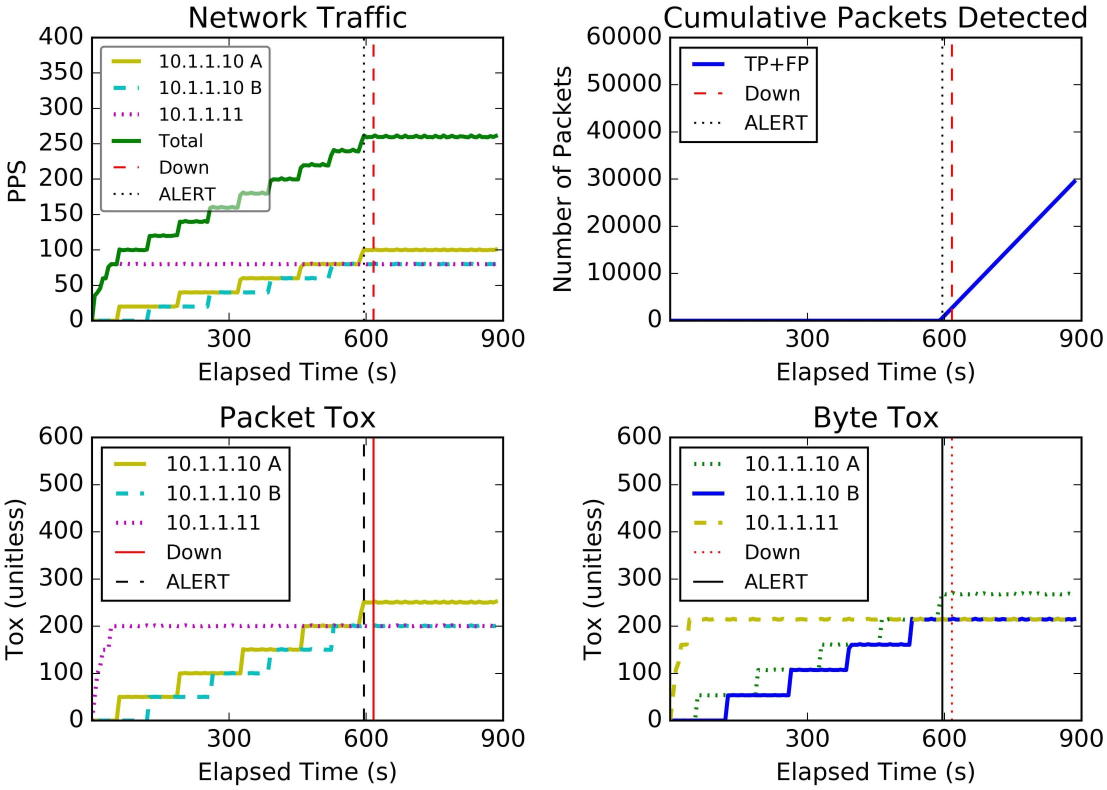

Fig. 5.7. BITMASK Profile gathering experiment results with pseudo-SOM supplied profile bitmask.

The toxicity figures illustrate the real benefit of this model. The benign subflow on the attack target does not surpass the alert threshold, and thus is not detected as malicious. The attack subflow, however, does, as is detected as malicious a result.

Table 5.3.
Profile gathering performance indicators

	TP	TN	FP	FN	DR	P	A	F
WILDCARD	28864	118669	23092	1100	96.329	55.555	85.902	70.469
BITMASK	28870	141587	0	1062	96.452	100.0	99.381	98.194

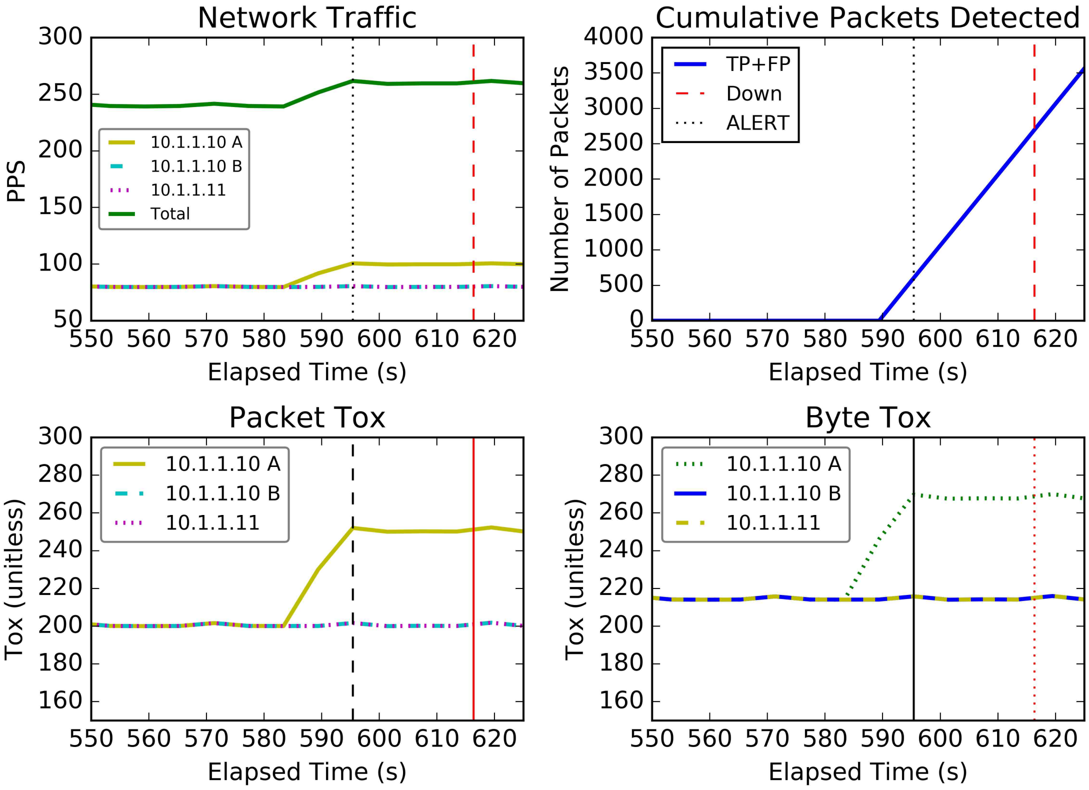

Fig. 5.8. BITMASK Profile gathering experiment results with pseudo-SOM supplied profile bitmask, detail.

This experiment's performance indicator results are given in table 5.3. As can be seen, the number of TP events is the same (the 6 packet difference is negligible) for the two detection profiles. The FN events with both profiles are associated with the 10 second detection time lag as a result of the AIS's polling period and toxicity deposit amount configuration. The great benefit of the BITMASK profile lies in the number of FP events. WILDCARD's wildcarded IP source field results in over 23000 FP events, giving a precision of only 56%. This inadequate performance is expected here, as it cannot distinguish subflows with such a wildcard when traffic levels per host are identical. Thus, WILDCARD cannot distinguish an attacking host from a benign host within a flow. BITMASK, however, can, thanks to its inclusion of the bitmasking pseudo-SOM. As a result, it creates 0 FP events and properly classifies WILDCARD's 23000 FPs as TN. This proper classification gives BITMASK perfect precision and an accuracy limited only by the polling period, greatly supporting the project's hypothesis.

Other works by Hong [30] and Jha and Archaya [26] have measured the same metrics for similar work, which is compared in table 5.4. The highest obtained F-scores are included here for comparison. It should be noted that the experiments vary here, and therefore the results of comparison should be taken lightly.

Table 5.4.
Related Work Performance Comparison

IDS	F-Score
Ais::BITMASK	98.2
DTA [30]	90.3
I3DS [26]	97.1

6. CONCLUSIONS AND FUTURE WORK

This work is intended to function as the laying of a foundational proof of concept for the proposed AIS network application system described in section 4. As demonstrated in section 5.3.2, when given a perfect profile, the AIS system can express perfect behavior, and thus is limited only by the strength of its detector generation method. Therefore, the hypothesis of this experimental proof of concept is well supported and should be extended.

The primary area of focus for future work should be on the development of the SOM module for the AIS network application. The development of this module would allow many of this project's assumptions to be removed and would allow the AIS to fully function as it was intended. The AIS network application is fully ready to accept network profiles from such an SOM module, so this development can focus on the SOM itself. The implementation details of the SOM module are not constrained by the AIS network application, and thus the SOM module can be implemented using any tool most suited to the task. The SOM generated profiles can be passed to the AIS network application through any usual application communication means, such as interprocess communication (IPC) [48]. If the developers of the SOM module choose to implement it in Python, minimal integration effort will be required, and the SOM can simply function as a separate thread/ process within the application.

Though the development of the SOM module will contribute mightily to the overall goals of the project, challenges remain that will limit its efficacy and accuracy. These challenges should be co-developed with the SOM module, as they are not logically connected and can be developed in parallel. These challenges include coordinating with the developers of the ofswitch13 module [38] to refine ns3::CsmaNetDevice modeling to solve the L3 Subnet Problem of section 5.2.3. Rectifying this problem will allow for a much more complex and realistic SDN network to be developed in NS-3.

Not only will this result in more applicable real-world simulations, but it will also provide for a better training environment for the SOM module. Development of this ofswitch13 module requires detailed knowledge of OpenFlow v1.3, general networking concepts, and strong C++ development skill. It is possible the developers of the module will solve this problem themselves, but it is not guaranteed.

The final workstream to be extended in the future is the development of realistic and complete NS-3 service request modeling, as discussed in section 5.2.1. The development of realistic μ values for destination terminal equipment packet service modeling would allow for the desired real-world effects of DDoS attacks on application servers to be modeled in NS-3 and a high analyzation resolution would result. This modeling must be able to account for load on the server, packet protocol, available server memory, etc. This sort of analyzation will be necessary before any potential AIS system is fully tested, so future developers will need this capability in testing. Development of this modeling feature requires detailed knowledge of NS-3, principles of queue modeling and random variables, and strong C++ development skill.

Future parallel development of these three workstreams is recommended, as the results of this paper demonstrate that a network-based AIS SDN network application is capable of detecting and mitigating DDoS attacks in progress in cloud computing environments.

REFERENCES

REFERENCES

[1] R. Van Der Meulen, *Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016*, 2017. Accessed: 6/26/2017. [Online]. Available: http://www.gartner.com/newsroom/id/3598917

[2] *2017 Study on Mobile IoT Application Security*, 2017. Accessed: 6/25/2017. [Online]. Available: https://media.scmagazine.com/documents/282/2017_study_mobile_and_iot_70394.pdf

[3] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[4] "Breaking down mirai: An iot ddos botnet analysis," Tech. Rep., 2017. Accessed: 6/25/2017. [Online]. Available: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

[5] T. Bowman, "Udp flood denial of service," Global Information Assurance Certification, Tech. Rep., 2001. Accessed: 7/1/2017. [Online]. Available: https://www.giac.org/paper/gcih/206/udp-flood-denial-service/101057

[6] J. Touch, "Tcp control block interdependence," Network Working Group, Tech. Rep., 1997.

[7] J. Postel, *Transmission Control Protocol*, RFC 793, 1981. Accessed: 6/25/2017. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt

[8] W. M. Eddy, "Tcp syn flooding attacks and common mitigations," Tech. Rep., 2007.

[9] D. Senie and P. Ferguson, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," Tech. Rep., 1998.

[10] W. Stallings and L. Brown, *Denial of Service*. Pearson Education Inc., 2008.

[11] S. Young and D. Aitel, *The Hacker's Handbook*. Auerbach, 2004.

[12] O. Reams, "Distributed denial of service (ddos) network-based detection," U.S. Patent 11 294 979, 2005. Accessed: 6/27/2017. [Online]. Available: https://www.google.com/patents/US20070130619

[13] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[14] O. S. Specification, "Version 1.3. 2 (wire protocol 0x04)," Tech. Rep., 2013.

[15] K. Hwang, G. C. Fox, and J. J. Dongarra, *Distributed System Models and Enabling Technologies.* Morgan Kaufman, 2012.

[16] S. A. Hofmeyr and S. A. Forrest, “Architecture for an artificial immune system,” *Evolutionary Computation*, vol. 8, pp. 443–473, 2000.

[17] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, “Self-nonself discrimination in a computer,” in *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium.* Ieee, 1994, pp. 202–212.

[18] J. Kim and P. J. Bentley, “An evaluation of negative selection in an artificial immune system for network intrusion detection,” in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation.* Morgan Kaufmann Publishers Inc., 2001, pp. 1330–1337.

[19] F. A. González and D. Dasgupta, “Anomaly detection using real-valued negative selection,” *Genetic Programming and Evolvable Machines*, vol. 4, no. 4, pp. 383–403, 2003.

[20] Z. Ji and D. Dasgupta, “Real-valued negative selection algorithm with variable-sized detectors,” in *Genetic and Evolutionary Computation–GECCO 2004.* Springer, 2004, pp. 287–298.

[21] Z. Ji and D. Dasgupta, “V-detector: An efficient negative selection algorithm with probably adequate detector coverage,” *Information sciences*, vol. 179, no. 10, pp. 1390–1406, 2009.

[22] P. Saurabh and B. Verma, “An efficient proactive artificial immune system based anomaly detection and prevention system,” *Expert Systems with Applications*, vol. 60, pp. 311–320, 2016.

[23] T. Kohonen, “The self-organizing map,” *Neurocomputing*, vol. 21, no. 1, pp. 1–6, 1998.

[24] *What is NS-3*, 2017. Accessed: 6/24/2017. [Online]. Available: http://www.nsnam.org

[25] M. Krasnyansky, “Universal tap/tun device driver,” The Linux Foundation, Tech. Rep., 2000.

[26] M. Jha and R. Acharya, “An immune inspired unsupervised intrusion detection system for detection of novel attacks,” in *Intelligence and Security Informatics (ISI), 2016 IEEE Conference.* IEEE, 2016, pp. 292–297.

[27] L. E. Baum and T. Petrie, “Statistical inference for probabilistic functions of finite state markov chains,” *Ann. Math. Statist.*, vol. 37, no. 6, pp. 1554–1563, 12 1966. Accessed 6/25/2017. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177699147

[28] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, *KDD Cup 1999 Data*, 1999. Accessed: 7/7/2017. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[29] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, “Cost-based modeling for fraud and intrusion detection: Results from the jam project,” in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX’00. Proceedings*, vol. 2. IEEE, 2000, pp. 130–144.

[30] L. V. Hong, “DNS traffic analysis for network-based malware detection,” Master’s thesis, Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, Asmussens Alle, Building 305, DK-2800 Kgs. Lyngby, Denmark, 2012.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. MIT Press, 2009.

[32] R. Braga, E. Mota, and A. Passito, “Lightweight ddos flooding attack detection using nox/openflow,” in *Local Computer Networks (LCN), 2010 IEEE 35th Conference*. IEEE, 2010, pp. 408–415.

[33] *NOX Network Control Platform*, 2017. Accessed: 7/4/2017. [Online]. Available: https://github.com/noxrepo/nox

[34] S. M. Mousavi and M. St-Hilaire, “Early detection of ddos attacks against sdn controllers,” in *Computing, Networking and Communications (ICNC), 2015 International Conference*. IEEE, 2015, pp. 77–81.

[35] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, “A sdn-oriented ddos blocking scheme for botnet-based attacks,” in *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conference*. IEEE, 2014, pp. 63–68.

[36] *The POX Controller*, 2017. Accessed: 7/4/2017. [Online]. Available: https://github.com/noxrepo/pox

[37] *Mininet: An Instant Virtual Network on your Laptop (or other PC)*, 2017. Accessed: 7/4/2017. [Online]. Available: http://mininet.org/

[38] L. J. Chaves, “Openflow 1.3 module documentation,” University of Campinas, Tech. Rep., 2017.

[39] *Floodlight SDN Controller*, 2017. Accessed: 6/25/2017. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[40] A. Leon-Garcia and I. Widjaja, *End-to-End Digital Services*. McGraw-Hill, 2004.

[41] *IEEE 802.3 Ethernet Working Group*, 2017. Accessed 6/25/2017. [Online]. Available: http://www.ieee802.org/3/

[42] *Python Programming Language*, 2017. Accessed: 6/25/2017). [Online]. Available: https://www.python.org/

[43] A. Leff and J. T. Rayfield, “Web-application development using the model/view/controller design pattern,” in *Enterprise Distributed Object Computing Conference, 2001. EDOC’01. Proceedings. Fifth IEEE International*. IEEE, 2001, pp. 118–127.

[44] D. Crockford, “The application/json media type for javascript object notation (json),” Network Working Group, Tech. Rep., 2006.

[45] A. Leon-Garcia and I. Widjaja, *Delay and Loss Performance.* McGraw-Hill, 2004.

[46] *ns3::CsmaNetDevice Class Reference*, 2017. Accessed: 7/1/2017. [Online]. Available: https://www.nsnam.org/doxygen/classns3_1_1_csma_net_device.html

[47] "The open group base specifications issue 7," The Open Group and IEEE, Tech. Rep., 2016.

[48] L. Lamport, "On interprocess communication," *Distributed computing*, vol. 1, no. 2, pp. 86–101, 1986.