Fall 2017

Predicting the impact of data corruption on the operation of cyber-physical systems

Erik David Burgdorf

Department:

PREDICTING THE IMPACT OF DATA CORRUPTION ON THE OPERATION OF CYBER-PHYSICAL SYSTEMS

by

ERIK DAVID BURGDORF

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

2017

Approved by

Sahra Sedigh Sarvestani, Advisor
Ali R. Hurson
Egemen K. Çetinkaya

ABSTRACT

Cyber-physical systems, where computing and communication are used to fortify and streamline the operation of a physical infrastructure, now comprise the foundation of much of modern critical infrastructure. These systems are typically large in scale and highly interconnected, and span application domains from power and water distribution to autonomous vehicle control and collaborative robotics. Intelligent decision support in these systems is heavily reliant on the availability of sufficient and sufficiently correct data. Failure or malfunction of these systems can have devastating consequences in terms of public safety, financial losses, or both.

The research described in this thesis aims to predict the impact of loss or corruption of data on operation of a cyber-physical system. Given knowledge of the respective physical and functional topologies of the system, information exchange is abstracted as a graph of interconnected data processing nodes. A model is created with each node modeled as a stochastic colored Petri net. Populated with information about the reliability of measurement, communication, storage, and processing devices, the Petri net model enables estimation of the fraction of the node's data that will be lost or corrupted.

A determination is made of each node's criticality, based on the consequences of its failure on overall system-level operation, using field data or simulation. The measure of data corruption impact at a given node is the product of the two aforementioned metrics: i) the extent of data corruption expected at the node and ii) its criticality. The proposed approach can enable informed decisions about targeted investments in hardening of cyber-physical system, specifically to mitigate the effects of corruption or loss of data.

ACKNOWLEDGMENTS

I would like to take this opportunity thank my thesis and research advisor, Dr. Sahra Sedigh Sarvestani. Her counsel during the decision process leading me to undertake completion of this research is highly valued. It is not an understatement to say that my completion of this program is due, in part, to her advocacy for and support of distance programs. Thanks are also due to the other members of my Committee, Dr. Ali Hurson and Dr. Egemen Çetinkaya. They both offered much of their time.

I would also like to acknowledge colleagues in our research group for their assistance, especially Mark Woodard for spending hours discussing model design, Koosha Marashi for his PSAT expertise and Isam Alobaidi for helping me get started.

Finally, but in no way least, I want to thank my wife, Lynn. Without your love, support and encouragement I would not have finished what was started long ago. I must also thank you for being the best tandem bicycle stoker I know. I have enjoyed every mile we have pedaled together. Thank you!

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

SECTION

1. INTRODUCTION

Many modern critical systems, ranging in scale from large distributed infrastructure services such as distribution networks for water or electrical power are no longer comprised of only traditional physical devices. In these *cyber-physical* critical infrastructure systems, the cyber components include sensors deployed throughout the system to collect real-time data. The data collected is then transmitted over communication lines to decision support components that analyze the data and determine configurations that will optimize aspects of systems operation. The configurations are then transmitted over communication lines to control devices where adjustments to systems operation are actuated. When operating correctly, cyber-physical systems (CPSs) are expected to deliver more efficient service, and often extended capability and features.

These improvements, however, come at a cost. Service delivery in a CPS is contingent on the availability of sufficient and sufficiently correct data. If data required for decision support is corrupted or missing, component- and/or system level-failures could result in catastrophic or even fatal consequences. The goal of the research described in this thesis is to quantify the effect of data loss or corruption on service delivery by a CPS.

Motivating examples for this work include past CPS failures caused by missing or corrupted data. The European Space Agency's ExoMars Schiaparelli Mars Lander was lost due to a transient data corruption event [3]. During descent, the lander's inertial measurement unit produced erroneous data for approximately one second. This erroneous data caused the navigation system to estimate a negative altitude, leading to premature release of the parachute at a height of 3.7 km. The lander plummeted to the surface, impacting at an estimated velocity of 150 m/s. The craft was destroyed on impact.

Another motivating example is the crash of Air France flight 447, which is believed to have been triggered by corrupt data reported from an iced-over pitot tube (the device providing airspeed data) [4]. The loss of data and unavailability of airspeed information caused the autopilot to disconnect. The flight crew had to take over control of the aircraft. With conflicting and incomplete airspeed data, the flight crew became confused and made decisions leading to eventual crash of the aircraft and loss of 228 lives.

As noted above, data integrity issues affecting large-scale cyber-physical infrastructure systems have the potential to affect many individuals and have significant economic impact. In July 2012, two widespread blackouts in India left over 600 million individuals - approximately 10% of the worlds population at that time - without power. Unreliable data was identified as one of the causes of these outages. A post-event inquiry committee noted the need for data reliability and recommended fortification of the communication networks responsible for delivery of data to the load dispatch centers [5].

The potential for data corruption in CPSs has been recognized for some time. Voas raised this concern in 1997 [6]. He noted that many CPSs are systems-of-systems, constructed by integration of multiple commercial, off-the-shelf software application packages. CPS designers necessarily are limited in detailed knowledge of semantics of each component. Between complexity and opacity, the semantic dependencies within systems-of-systems are such that no single person can fully understand all details. There will likely be undiscovered incompatibilities, leading to data corruption [1]. Voas proposes a methodology, *interface propagation analysis*, where errors are deliberately injected into an operational system. Propagation of corrupted data is then monitored to determine impact. This methodology is often infeasible in practice, as it requires testing of full-scale in-use systems, or at a minimum, a duplicate test system. Crenshaw et al. propose what they

[1]The European Space Agency report notes this situation occurred in the Schiaparelli ExoMars Lander incident. Individual components did not fail; they operated correctly during the landing phase. It was the interaction of components and undiscovered gaps in specifications that resulted in the loss of the spacecraft.

denote as the *simplex reference model* to limit fault propagation in CPSs [7]. They too note that the integration of commercial off-the-shelf software into a CPS results in less than full understanding of system behavior. Full verification becomes impossible.

In his PhD dissertation, Behrens notes that the hardware errors currently observed in production systems could be considered inevitable, the rate of these errors is expected to increase in future hardware generations, and a non-negligible number of errors are uncorrectable by hardware techniques such as error correcting codes and manifest as application state corruptions [8]. These errors are potentially propagated to other components when a process experiencing state corruption sends corrupted data to an external entity, i.e., sends a corrupt message.

An understanding of the impact of missing or corrupt data has on cyber-physical system components is required in order to enable appropriate hardening measures for the most vulnerable and/or critical components. The objective of the research described in this thesis is to create and demonstrate a methodology that quantifies the impact of corrupt and missing data on the operation of CPSs. Previous studies have investigated the criticality of physical components, as well as the impact that individual components have on overall system service delivery. The original contribution of this thesis is a model that provides an a-priori relative measure of the impact of missing and corrupt data on service delivery from each cyber component. This model utilizes stochastic colored Petri nets to estimate the flow of corrupted and missing data across the components of a CPS, and utilizes this information in conjunction with estimates of physical component criticality to provide a measure of the system-level impact of data loss or corruption on service delivery. Understanding this impact can guide prioritization of hardening investments, leading to more robust CPSs with improved service delivery characteristics.

The remainder of this thesis is structured as follows: Section 2 provides an overview of foundational and related work. Section 3 describes the proposed approach. Section 4 illustrates the application of the proposed approach through a case study involving the IEEE 57-bus test system. Lastly, Section 5 provides concluding notes and describes areas for continuing research.

2. BACKGROUND AND RELATED WORK

Critical cyber-physical systems, by nature and definition need to be reliable, dependable systems. For the purposes of this thesis, dependability attributes are defined as described by Avizienis et al. [9]:

availability: readiness for correct service.

reliability: continuity of correct service.

safety: absence of catastrophic consequences for the user(s) and the environment.

integrity: absence of improper system alterations.

maintainability: ability to undergo modifications and repairs.

As demonstrated by the motivating examples of the previous section, dependable system operation requires sufficient and sufficiently correct data in order to deliver services at the expected levels. Unfortunately, despite the attention paid to reliability during the design phases of the system and its components, it is impossible to guarantee 100%-reliable systems. As systems grow in scope and complexity, software and hardware errors become unavoidable. Hardware errors are observed surprisingly often and their rates are expected to only increase in successive hardware generations. Furthermore, a non-negligible number of errors are uncorrectable by current hardware techniques, such as error correcting codes, and manifest themselves as application state corruptions [8].

- A study by Schroeder et. al found error rates in deployed commodity servers orders of magnitude higher than previously reported, with 25,000 to 70,000 errors per billion device hours per Mbit. More than 8% of DIMMS were affected by errors per year [10].

- Hwang et. al studied DRAM errors from a diverse range of production systems collecting almost 300 terabyte-years of data. Analysis indicated that almost a third of memory banks showed signs of hard errors [11].
- Intel researchers note the reliability per bit tends to decrease by 8% each successive processor generation [12], [13].

These failures can have significant real-world impact as illustrated in the following anecdotal examples of failures [8]:

- Hardware faults corrupted the state of processes in some instances of Google's Chubby locking service resulting in at least one complete service failure.
- In 2008, a single flipped bit in a single machine propagated via messages to other processes, causing an 8-hour outage of Amazon S3 service in the USA and Europe.
- When under high load, faulty hardware corrupted the payload of Amazon S3 user message during 2008 and 2011.
- Magnolia, a social bookmarking website, had a major data corruption episode, losing all user data; an event that led to the company's closing.
- State corruption at the CPU level is a common occurrence in the Google Mesa data warehouse system at the large scale in which it operates. Ensuring state integrity becomes the responsibility of application developers who must add additional code to check assertions, checksum results, and execute consistency checks against replicated data [14].

Data corruption can also occur when malicious actors are present. Encryption, message authentication codes, digital signatures and cryptographic hash functions are used to protect sensitive information, authenticate users and detect message corruption [15]. Malicious actions are not limited to communication channels. Researchers have demonstrated

that parasitic effects in dynamic RAM can be exploited to change contents of neighboring memory cells [16]. A proof-of-concept exploit was developed to demonstrate the feasibility of this attack for off-the-shelf systems and further demonstrate the ineffectiveness of existing countermeasures.

In addition to hardware errors and malicious actors, increasing system complexity can cause potential sources of data corruption to go unnoticed. Attempts at reducing the resulting errors include *interface propagation analysis* and the *simplex reference model*, which were described in Section 1. In the latter, redundant controllers, with different operating characteristics, are utilized, providing operational diversity. An *unreliable component* provides full features and aggressive behavior, while a *trustworthy component* provides limited features with predictable, safe behavior. A third component, a *checker*, evaluates inputs from the unreliable component and trustworthy component, selecting and propagating the input that is expected to maintain safe operation. In this manner corrupted data is not allowed to propagate and cause the CPS to operate outside of defined safe operational limits. This approach comes with additional costs of procuring and integrating the duplicate and arbitrating components.

Many techniques have been proposed for modeling the reliability of combined hardware/software systems. As an example, work by Friedman et al. develops extensive statistical procedures to calculate reliability measures for hardware and software components [17]. Despite this general activity, as of the publication date of this thesis, no studies outside of our research group have aimed to specifically capture the effect of data corruption on dependability attributes of a system.

Petri nets, and their derivatives, have been used as a basis for system reliability models. The work most closely related to this thesis is presented in [18], which attempts to illuminate dependencies between electrical and cyber infrastructures and quantify the impact of data loss resulting from a malicious attack. The Petri nets described in their work account for some missing data, however, the impact of corrupt and missing data is not

the focus of the study. Malhotra et al. use Petri nets to model system dependability [19]. They propose a methodology to convert fault trees to general stochastic Petri nets (GSPN) and stochastic reward nets (SRN), allowing for analysis of system dependability attributes. Their work does not directly address data corruption. Chen et al. investigate the use of Petri nets to model coordinated attacks on an electrical smart grid [20]. These attacks include intentional, malicious data corruption or blocking of communication paths, which results in data loss. Their work shows the sequence of actions during attacks, with the aim of gaining an understanding of attack vectors. Propagation of corrupt data or ripple effects of data loss are not directly addressed.

Understanding of component criticality for CPSs and determination of the impact of individual component failures on service delivery are recognized needs. Sha et al. were among the first to note the pressing need for theory and tools that facilitate understanding of dependency relationships between components and the effects of these dependencies on service delivery [21].

Failure dynamics within single, standalone networks have been the subject of extensive study. Systems today consist of coupled networks, whose failure dynamics differ from those of individual networks. Among these differences is the increased vulnerability of coupled networks to random failures [22].

A mechanism to quantify the impact of cyber components within a CPS is a first step. Marashi et al. propose such a model, calculating two measures of CPS component dependability; criticality (impact to overall service provision when a component fails) and fragility (sensitivity of this component to failure of other components) [23]. This model quantifies the interdependence of CPS cyber and physical components, and quantifying the effect of fault propagation paths in the face of disruptive events. Woodard proposes a conceptual CPN data corruption model [24]. This model facilitates the evaluation of data corruption extent at any given time and allows for investigation of the impact data corruption has on service delivery.

This work extends the foundational work of Woodard and Marashi. The original research contribution of this work is a model that quantifies and predicts the impact corrupt and missing data have on cyber-physical data processing elements and the resultant impact to overall system service delivery. The proposed approach and metric can enable informed decision making in the hardening decision process, providing information as to which data processing elements within a cyber-physical system, that fail due to missing or corrupt data, have the most impact on overall system service delivery. Conversely, the information provided by this approach informs decision makers as to those data processing elements where missing and corrupt data issues have lower impact on service delivery. When limited resources are available the proposed model can be used as an aid to applying resources where greatest impact will be realized.

2.1. MODELS

Real world physical systems components operate concurrently and in a non-deterministic manner. Designing reliability into a cyber physical systems is a challenge and non-trivial task. For example, in a system of even minor complexity, when communicated messages are lost, process scheduling and timing of physical events are taken into account the number of execution states becomes quite large - larger than reasonable for human designers to understand all interaction patterns, including the impact of corrupted or missing data on system operation. For many critical systems; vehicle control systems, life support systems, nuclear power plant control systems, etc. reliability must be understood and designed in to the system from the beginning. Further, it is impossible to inject faults and monitor failures in many operating cyber-physical systems. Disruption caused by testing is not an option as these systems provide critical services.

Models have become a preferred method to address these design challenges. Developing models allows for a designer to gain 1) *insight* into the operation of complex, concurrent systems; 2) understand *completeness* of the design and 3) confidence in the

design *correctness*. Petri nets and derivatives; colored Petri nets, stochastic Petri nets; have been used as modeling environments that can aid in meeting these design challenges. Two examples of use of Petri nets to model business processes are:

- Ericsson Telebit used CPNs to aid in development of the communication protocol Edge Route Discovery Protocol (ERDP) [25]. Modeling, simulation and state space analysis identified several issues and errors in the design prior to finalization.
- Huang et. al [26] used Colored Petri Nets to develop models and algorithms for tracing the entire life of manufactured products.

2.2. INFORMAL DEFINITION PETRI NET

Petri nets are directed bi-partite graphs consisting of two mutually exclusive node classes, labeled as either a *Place* or a *Transition*. *Arcs* connect places to transitions and vice-versa. It is invalid for an arc to connect two places or two transitions. Each place may contain one or more *Tokens*.

In a standard PN, tokens are untyped and no distinction is made between individual tokens. Places contain discrete numbers of tokens. The set of tokens held in a particular place is the *Marking* of that place. A distribution of tokens across all places is denoted as a *Marking* of the Petri net. Transitions are enabled when sufficient tokens exist in all of the transition's input arcs. When enabled a transition may *Fire* consuming input tokens and producing tokens on output arcs. In a standard PN execution is nondeterministic when

multiple transitions are enabled simultaneously. The *Initial Marking* of the PN is specified by assigning a number of tokens to Places as required by the model. Graphically, Colored Petri Nets are represented as follow:

- *Places* as ovals
- *Transitions* as rectangles
- *Arcs* by line segments.

Arrowheads indicate direction of token flow. Arcs connecting transitions to places are referred to as *Input Arcs*, arcs connecting places to transitions are referred to as *Output Arcs*. Figure 2.1 shows an example of a simple CPN.

Petri Net markings over time create a *State Space*. Analysis of the state space allows for understanding of system behavior and properties, such as state reachability.

2.3. PETRI NET DEFINITION

A basic Petri net, N, is defined as the tuple $N = (P, T, A, M_0)$ where:

- $P = \{p_1, p_2, ..., p_n\}$: a finite set of places
- $T = \{t_1, t_2, ..., t_n\}$: a finite set of transitions
- $P \bigcap T = \Phi$ and $T \bigcap P = \Phi$
- $A = \{a_1, a_2, ..., a_n\}$: a finite set of arcs
- $A \subseteq (PxT) \bigcup (PxT)$
- $M_0 = \{mp_1, mp_2, ..., mp_n\}$: the initial marking

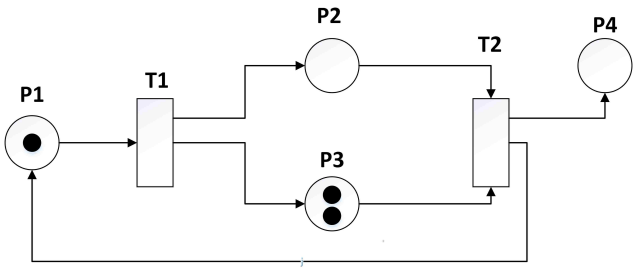

Figure 2.1. Simple Petri Net

2.4. COLORED PETRI NETS

An extension to basic Petri Net models is Colored Petri Nets (CPN). CPNs extend PNs by adding a type, *ColorSet*, with enumerated (possibly infinite) values, *Color* to the token specification. This allows for distinction among tokens and behavioral changes based on token type and value. The definition of Place is extended to include a list of permitted ColorSets allowed to be held as tokens in each Place. Arc definition is extended to include an *Arc Expression* function or functions. Type checking is then performed on input and output arcs. Arcs are enabled only when sufficient number of corresponding token types are available. A second extension, *Guard Expression* is also added. Guard Expressions evaluate input arc token types and values to produce a Boolean true or false value. If the evaluation results in a true condition the arc is enabled, if the evaluation evaluates to false the arc is disabled. If the Guard Expression evaluates to false, the transition is not enabled (irrespective of the number and type of input tokens). The Guard Expression concept enables multiple simultaneous arcs to connect the same Places. Behavior is determined at runtime based on token type and value. If there are multiple enabled transitions connecting two places, only one is selected using a stochastic process.

Examples of Petri include:

- Modeling of human and animal metabolic systems, [1]
- Design of urban traffic light control system, [2]

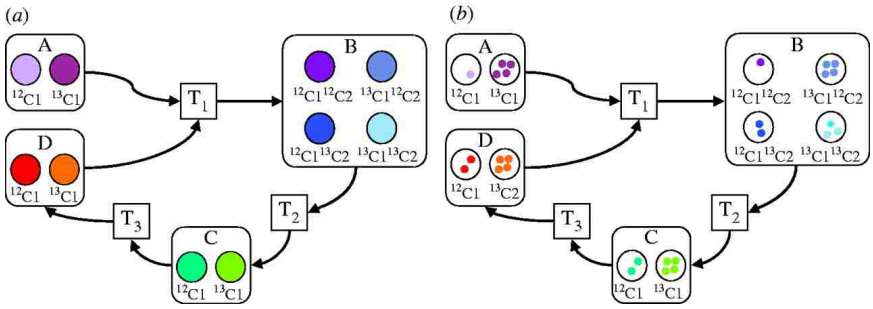

Figure 2.2. Colored Petri Net Used to Model Human Metabolic Systems [1]

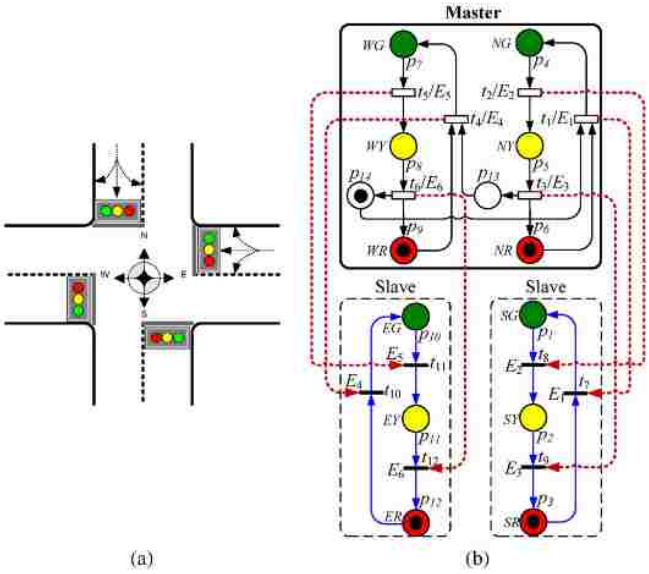

Figure 2.3. Colored Petri Net Used to Model Urban Traffic Control System [2]

2.5. COLORED PETRI NET DEFINITION

The Colored Petri net Definition includes definitions of the standard Petri net and adds:

A DTSCPN, N, is thus defined as the tuple $N = (P, T, A, \Sigma, F, C, E, G, M_0)$. Definition of $P, T, AandM_0$ are as in the original Petri net definition. The additional tuple elements are defined as:

- $\Sigma = \{c_1, c_2, ..., c_n\}$: a finite set of non-empty color sets (type)
- F = node function mapping A onto $(PxT) \bigcup (PxT)$

- C = color function mapping P into Σ
- G = guard function mapping T into expression: $\forall t \in T$, type $G(t) = BOOLEAN$
- E = arc expression mapping from A such that $\forall a \in A$, type $E(a) = C(p(a))$

2.6. DISCRETE TIME STOCHASTIC PETRI NETS

Molloy describes a further extension to Petri nets, discrete-time stochastic Petri nets (DTSPN) [27]. DTSPNs further extend Petri nets and colored Petri nets with specification of a non-zero firing probability for each enabled transition. The discrete time stochastic Petri net may be viewed as a standard Petri net where at each time step any number (including none) of enabled transitions may fire based on probabilities attached to transitions. The probability of an enabled transition firing is "memoryless," depending only on the probability specified and not on previous firing history or current time step. Discrete time stochastic Petri nets are thus Markovian in nature, the reachability graph of a DTSPN can be mapped to a Markov process. This work uses this Petri net derivative.

3. METHODOLOGY

3.1. CONTRIBUTION

The original research contributions of this work are twofold. First, architectural changes were made to a colored Petri net simulation model to increase modeling performance, provide programmatic control of transition firing probabilities, and enable interaction with external tools during simulations via a set of application programming interfaces. Second,a model is developed that quantifies the potential impact missing and corrupt data have on CPS data processing components and the resultant impact to service delivery. This metric provides a calculated relative measure of service delivery impact to each component, and provides a means to rank corrupt and missing data impact component by component. It is envisioned this metric will be used in the hardening decision making process, providing data identifying those areas of a system having most impact of service delivery when affected by missing and corrupt data and those areas of a system where missing and corrupt data have lower impact on service delivery. When limited resources are available the proposed model can be used as an aid to applying resources where greatest benefit will be realized.

This work extends two foundational research projects, those of Woodard and Marashi [24], [28], [23]. Consistent with their work, survivability is defined qualitatively as a system's ability to continuously deliver essential services and exhibiting failure resistance and graceful degradation. Two survivability quantitative metrics are seen in Figure 3.1, *Extent of Degradation* and *Rate of Degradation*. These metric will be calculated as:

- Extent of Degradation: $\delta = \max_{t_0 \leq t \leq t_d} |M(t_0) - M(t)|$
- Rate of Degradation: $\rho = \max_{t_0 \leq t \leq t_d} \left| \frac{dM(t)}{dt} \right|$

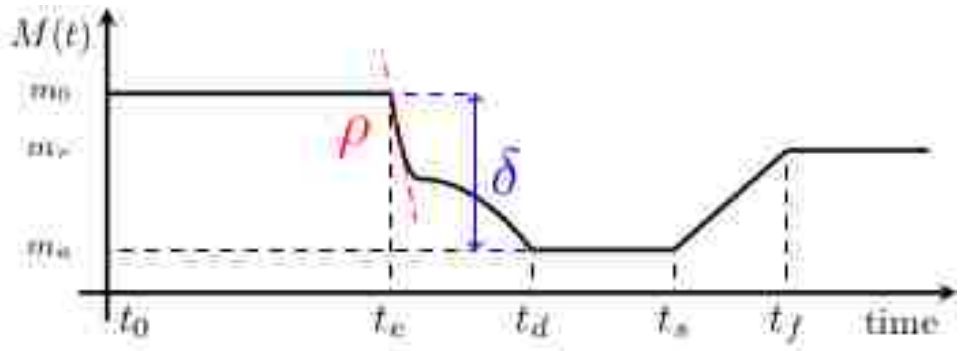

Figure 3.1. Survivability Metrics

This work extends two aspects of the Marashi and Woodard work. An idealized Petri net model was proposed to enable quantification of missing and corrupt data propagation. The idealized model implements an idealized 3x3 system model. This work extends this model from the described idealized 3x3 topology to allow topologies modeling the data processing element topology within a cyber-physical system. Second, a methodology was proposed in these works to calculate the **Criticality** of each component. Criticality is defined as a measure of the consequences of a component failure on overall service delivery. In the referenced work, to calculate criticality a set of failure cases is generated and simulated. For each failure case k, the highest level of service degradation is measured as δ_k. Component criticality, α_i, is calculated as the sum across all failure cases where component i fails as:

$$\alpha_i = \frac{1}{m} \sum_{k \in Q_i} \left(\overbrace{\frac{\delta_k}{\max\limits_{1 \le l \le m} \delta_l}}^{\text{first term}} \times \frac{\overbrace{\left.\frac{dM_k(t)}{dt}\right|_{t=t_i^{(k)}}}^{\text{second term}}}{\max\limits_{\forall t} \frac{dM_k(t)}{dt}} \times \frac{\overbrace{\left.\frac{d^2M_k(t)}{dt^2}\right|_{t=t_i^{(k)}}}^{\text{third term}}}{\max\limits_{\forall t} \frac{d^2M_k(t)}{dt^2}} \right) \tag{3.1}$$

The first term normalizes the amount of service delivery impact. The second term normalizes the rate of impact at the instant of component i's failure at time t during failure case k. The third term, the second derivative, is considered to be indicative of the immediate impact of component i's failure during the failure case.

This work extends this method and it's application to quantify the criticality of data processing component failures to service delivery. A high level view of this model is found in Figure 3.3. This figure shows data flows within and across the model components. Steps in this process are:

I. The *data processing topology* corresponding to the cyber-physical system topology is abstracted as a graph

II. An estimate of the *extent of data corruption* expected at each node is determined using a colored Petri net model

III. The set of *data processing component criticality* values are calculated

IV. The two data sets are provided as input to a calculation of *data corruption impact* for each cyber component.

3.1.1. Step 1: Topology Abstraction. The first step in the process is design of a colored Petri net model mapping the CPS cyber component topology into a colored Petri net topology. A standard CPN node is defined to model each cyber component. Figure 3.2 provides a graphical view of Petri net components within a node. Following standard Petri net nomenclature, places are denoted with ovals, transitions with rectangles and arcs as connecting arrows. Direction of data flow is indicating by the arrow direction with two-headed arrows indicating two-way data flow. Components within each node are:

- Sensor(s): one or more sensors providing local data
- Input: an entry point to receive data from connected nodes
- Detection and Cleansing: entities that detect and cleanse incoming data from sensors and inputs
- Database: a database holding a set of historical data values
- Processing: local processing of database entries which may produce corruption

This node is then connected to other nodes as defined by the cyber topology creating the complete CPN model of the CPS cyber components.

Rather than using fixed values for communication, sensor and processing failures, and to enable closer modeling of real-world devices, node-specific probability values are assigned to the following events:

- Probability of local sensor(s) producing corrupt data
- Probability of local sensor(s) not producing data when expected (i.e. missing value)
- Probability of local processing corrupting entry in node database
- Probability of local processing dropping data from node database (i.e. creating a missing value)
- Probability of inter-node communication producing corrupt data
- Probability of inter-node communication not producing data (i.e. producing a missing value)

Node definitions include optional, configurable mitigation capabilities. If enabled, corrupt and missing data are cleansed at specified probabilities.

Global parameters specified include number of simulation steps (time) and the number of simulation iterations to execute.

3.1.2. Step 2: Data Corruption Extent Estimation. Analysis is performed on simulation output to collect data on amount of corrupt and missing data per execution step per node as well as the contiguous step sequence lengths where corrupted and missing data exceed specified limits. Figure 3.4 shows a graphical representation of this data for the decision support node. [1] Output of this step, $\mathbb{A}$ used in the final step is the average

[1] data shown reflect results from running multiple simulations, with 25,000 steps per simulation run.

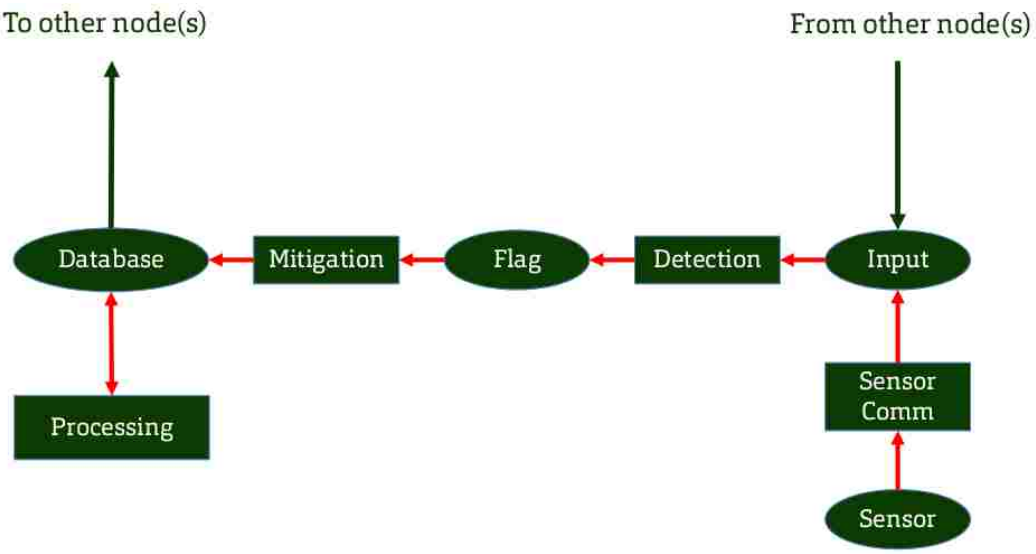

Figure 3.2. CPN Node Structure

percentage of corrupt and missing data within the node database.

$$\forall\ n\ in\ N\ :\ \{\text{DTSCPN nodes}\} \tag{3.2}$$

$$A_n = \frac{\sum_{steps} \%\ db\ corruption_n}{n} \tag{3.3}$$

For purposes of this work average corruption percent is calculated across all simulation steps. [2]

Figure 3.5 shows a distribution of the number of consecutive simulation steps where corrupted and missing data exceed a defined threshold for the device F_6. This data can provide insights regarding how much missing and corrupt data must be tolerated in an individual node while continuing to provide service. This data is provided to demonstrate that shorter runs of corrupted data over the defined threshold occur more frequently than longer runs of corrupted data. The model presented her provides this data however full analysis of the data is outside the scope of this work 5.

[2]This work includes capture of addition model state space data. While note used in analysis described in this work it is available for future research. Refer to Section 5 for a description.

3.1.3. Step 3: Data Processing Component Criticality Calculation. The second process step is to calculate an aggregate criticality of cyber components, *Data Processing Component Criticality*. The concept of criticality was introduced by Marashi [23]. Criticality can be informally expressed as a measure of how much a single component affects system survivability. Marashi provides a formal definition of criticality and methodology used to calculate criticality for a single physical component. Multiple physical components can be affected and controlled by a single cyber component. This work introduces the concept of aggregate criticality. Data processing component criticality is, in essence, a measure of the cascading criticality of all physical components directly connected to a specific data processing component. For purposes of this model it is assumed that failure of a cyber component results in failure of those physical components directly attached.

Data processing component criticality is calculated as:

$$\forall p \in P : \{\text{physical components}\} \tag{3.4}$$

$$\forall c \in C : \{\text{cyber components}\} \tag{3.5}$$

$$\alpha_i = \text{criticality of physical component } i \tag{3.6}$$

$$d_c = \{\text{physical components directly affected by} c\} \tag{3.7}$$

$$criticality(C_c) = \frac{1}{|d_c|} \sum_{\forall d_c} \alpha_i \tag{3.8}$$

$$\tag{3.9}$$

3.1.4. Step 4: Impact Calculation. The last step to combine the data processing component criticality from Step 3 with corresponding corrupt and missing data measured from Step 2. Impact is calculated as:

$$\forall c \in C : C = \{\text{cyber components}\} \tag{3.10}$$

$$Impact_c = Criticality_c * A_c \tag{3.11}$$

3.2. SIMULATION PERFORMANCE AND INTEGRATION

The foundational colored Petri net simulation code was developed to demonstrate and validate this particular approach to modeling data corruption behaviors. Performance concerns were not included in the original design goals. To enable simulation of large and complex models it was necessary to analyze performance inhibitors and identify code changes leading to improved simulation performance. Analysis of the simulation environment and model architecture identified several areas where changes provide increased simulation performance.

3.2.1. Design Review - Performance Inhibitors. While reviewing foundational work, it was noted that CPN simulation performance was an inhibitor to modeling of large, complex systems. Analysis of the CPN model and architecture provided opportunity for architectural changes that would provide increased performance and add additional capabilities while maintaining existing model semantics.

Individual nodes in the original model design consist of four places, twenty nine transitions and ninety two arcs Figure 3.6. For each simulation step a marking calculation is executed. This calculation consists of:

- Boolean guard functions are evaluated to determine is transition is enabled.
- Possible firing modes for each enabled transition are calculated by iterating over each combination of input token(s).

When the step executes a random firing is selected from the set of firing modes for each transition

Linux performance tools and the Python pStat module were used to characterize performance and identify potential areas for performance enhancement. This characterization indicated calculation of the marking iterations as computationally expensive and an area where significant performance improvements could be realized. Table 3.1 shows captured

data from the original node structure. Analysis of this data indicated significant time spent in Python functions calculating possible firing modes. Simplification of node architecture to reduce the number of transitions and arcs would reduce the number and size of potential firing sets and have the desired performance enhancements.

Further review of the original node structure showed that multiple, mutually exclusive transition guards were used to model probabilities of data corruption and cleansing. For example, the process of communicated a valid token required three transitions; one for the case where the token is successfully transmitted, one for the case where the token is corrupted and one for the case the communication failed and the token is missing. Duplicate arcs are required for each of these transition. Arc duplication resulted in duplicate iterations with a corresponding performance impact.

3.2.2. Architectural Changes for Performance. A node simplification approach was chosen to eliminate the use of per-transition guards and utilize Python language functions. This approach resulted in a node structure with reduced number of elements; four places, five transitions and eleven arcs [Figure 3.7].

Increased performance is also realized by migrating from use of transition guards to user defined Python functions. This eliminates evaluating each guard function to determine if enabled. The prior approach had multiple transitions with mutually exclusive guards. This architecture necessitated enumeration of large number of possible firings. Using a single arc with user defined callable Python function logic significantly reduces the number of marking enumerations. Selection logic is pushed in to the user-written Python code. More complex logic is possible than with the transition guard approach. The programmatic approach also enables future work (see Section 5) allowing dynamic failure probabilities, communication and integration with other simulation tools

Tables 3.2 shows data from the re-architected node structure. Performance is significantly improved. It is noted that changes result in greatly decreased number of library calls. Overall execution time is also reduced. The same configuration and input files were provided to both scenarios.

The following improvements are realized: [3]

- Average runtime [4] reduced 47% (Table 3.3)
- Total function calls are reduced from 32,627,180,048 to 4,107,468,485.
- Procedures associated with marking enumeration dominate the original node profiling data and not the simplified node.

Table 3.1. pStat Data: Original Node

Calls	Total Time	Source File	Function
1323597000	3197.956	data.py	__add__
2862278160	1495.032	data.py	cross
924429896	1315.219	nets.py	bind
2021316504	1251.976	data.py	__init__
11828344	1092.502	nets.py	modes
918670448	1065.950	nets.py	_check
915100035	793.465	built-in	_functools.reduce
2647194000	700.647	data.py	dict
1218004298	672.161	nets.py	__init__
22870400	633.780	nets.py	<listcomp>

Table 3.3 describes run time improvements following modifications.

3.2.3. Integration. Architectural and code changes were implemented allowing for communication, exchange of control, and exchange of state information between the colored Petri net simulation code and external applications. These integration additions enable an external application to monitor simulation status, query the colored Petri net model markings, modify the colored Petri net model markings, and to control simulation

[3]Simulation environment: Hardware: 3.40GHZ Intel i5-3570K CPU, 8GB DDR2 memory. Operating System: CentOS 7 Linux (kernel version 3.10.0). Platform: Python 3.5. Data capture from 5 runs, 1000 steps per run.

[4]Runtime measurements recorded under profiling environment

Table 3.2. pStat Data: Revised Node

Calls	Total Time	Source File	Function
65949625	169.710	nets.py	_check
134599130	150.812	nets.py	bind
127428690	140.599	data.py	_add
72024565	133.213	nets.py	check
101373765	115.053	data.py	iterate
173463455	114.657	built-in	hasattr
213798470	109.670	nets.py	__init__
152628510	102.378	hashables.py	__setitem__
2699880	91.940	nets.py	modes
213805745	87.310	nets.py	__setattr__

step execution. The interface is enabled via a command line parameter and based on TCP/IP V4 socket interfaces. TCP/IP socket interface was selected due to the wide level of support for exploitation of this programming model. For example, the MATLAB-based Power System Analysis Toolbox (PSAT) could be extended to connect to, control, and query a CPN model markings, using the returned data to adjust power flow parameters based on levels of data corruption.

The external application is configured to listen for connections on a specified port. If enabled, the CPN simulation attempts to bind to this port and proceeds to communicate with the external application via a series of message flows. Figure 3.8 describes this content and flow of messages.

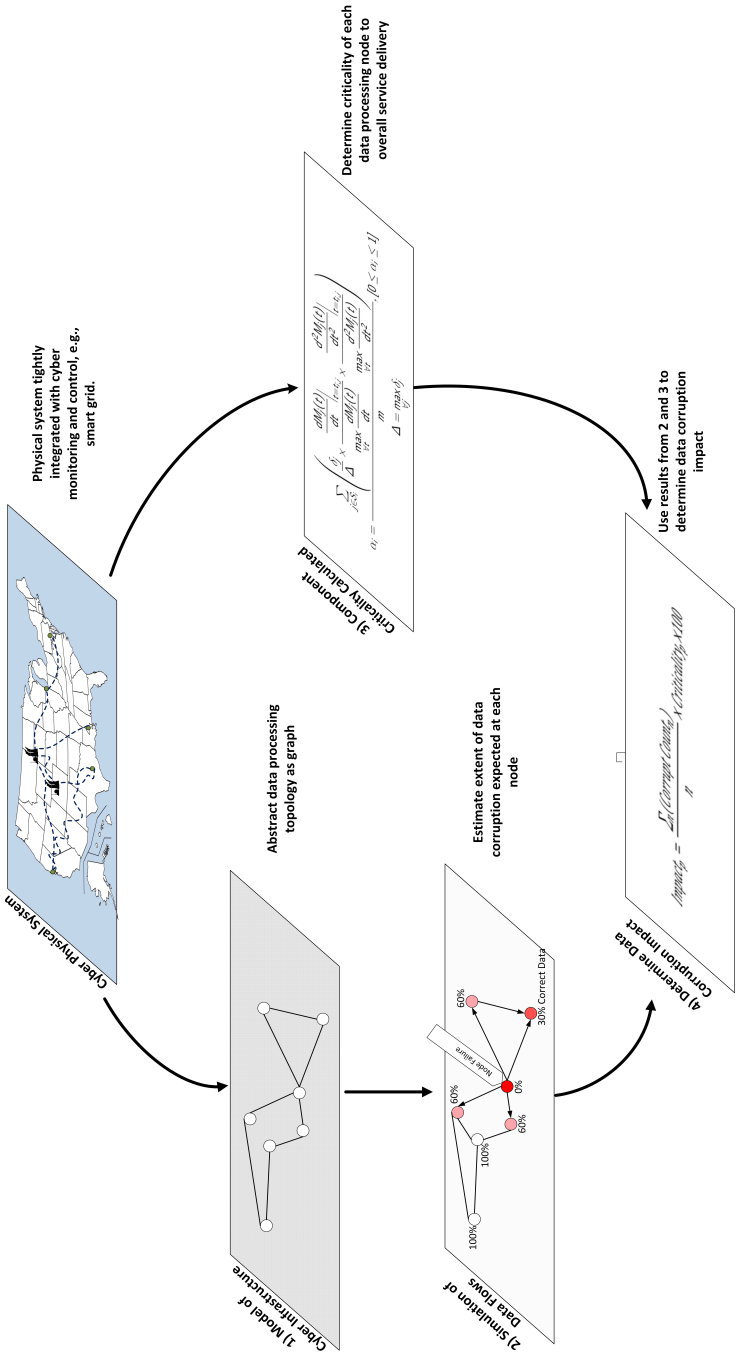

Figure 3.3. Model Overview

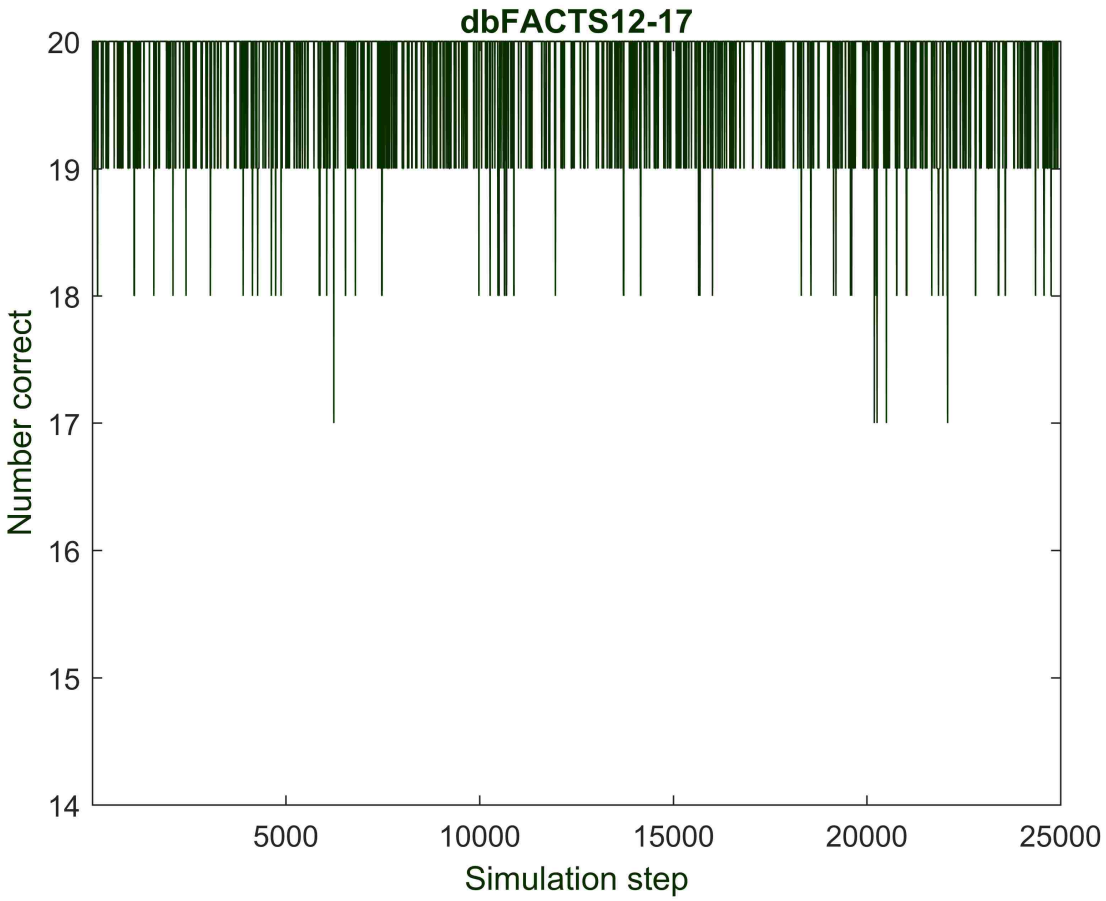

Figure 3.4. FACTS Device F_6 (l_{12-17}) Database State - Single Run

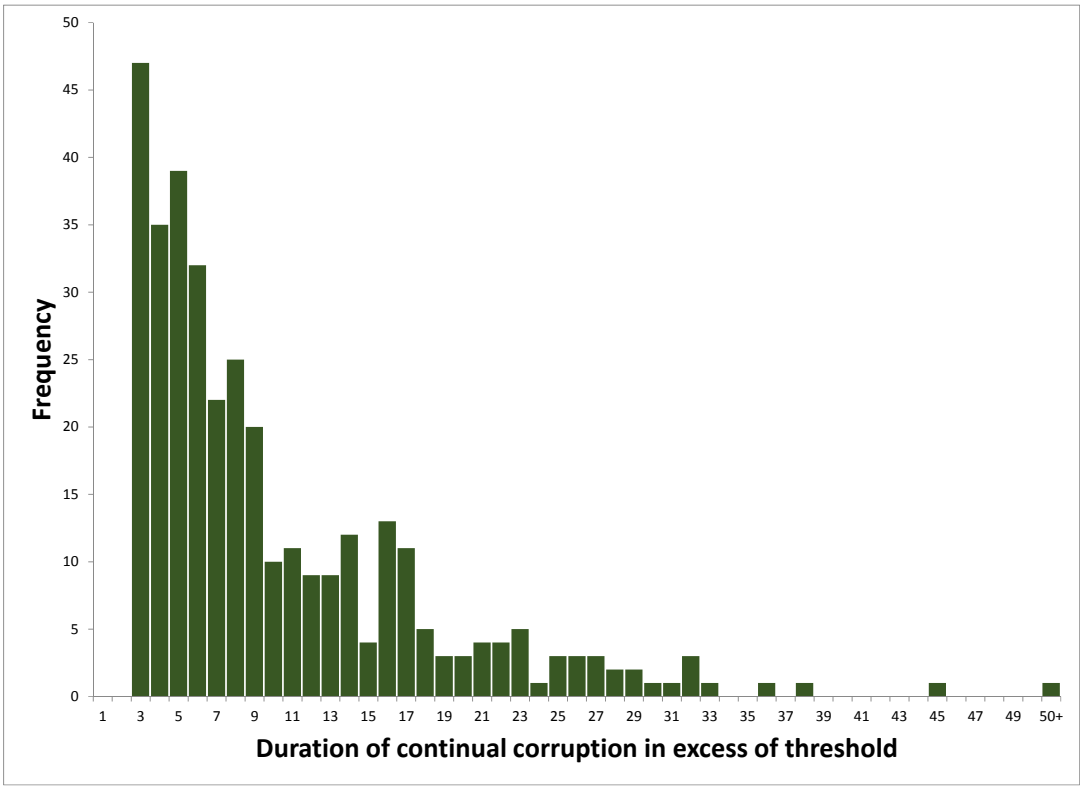

Figure 3.5. FACTS Device F_6 (l_{12-17}) Consecutive Steps Over Threshold - Single Run

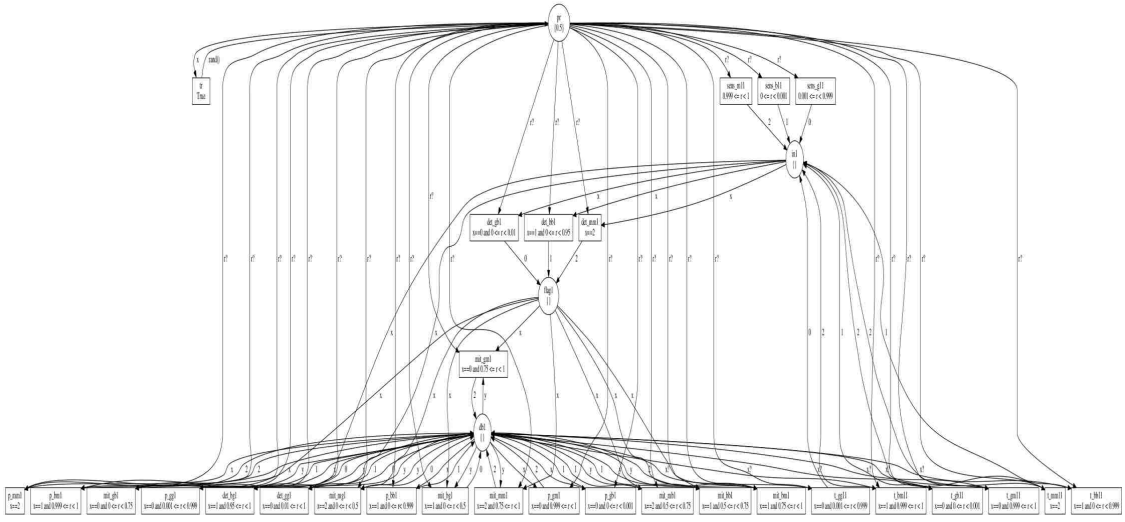

Figure 3.6. Original Node Structure

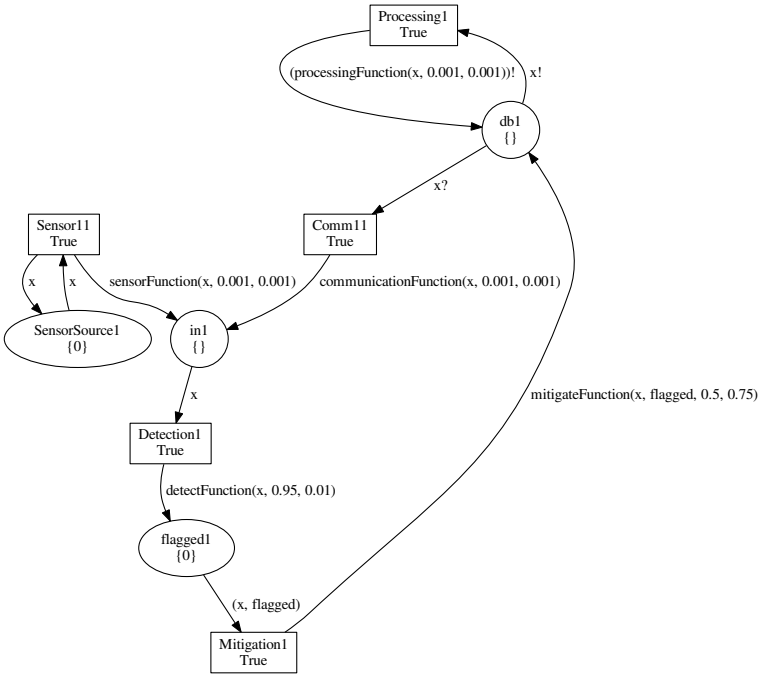

Figure 3.7. Simplified Node Structure

Table 3.3. Run Time in Seconds

Run	Original	Simplified
1	350.48	176.81
2	327.93	176.98
3	321.92	178.07
4	324.61	177.48
5	320.16	177.19
6	322.81	178.17
7	348.03	175.89
8	328.75	177.15
9	347.65	176.31
10	339.48	174.97
11	329.82	176.16
12	340.65	177.05
13	329.36	175.15
14	337.34	174.99
15	350.30	175.97
16	341.09	175.91
17	329.57	175.48
18	339.39	176.15
19	345.64	173.92
20	344.87	174.83
21	339.39	176.12
22	324.67	175.28
23	330.67	175.26
24	324.07	179.67
25	324.23	182.62
26	331.50	179.59
27	350.09	183.12
28	339.34	186.14
29	327.08	188.29
30	331.12	183.11
Average	334.73	177.79

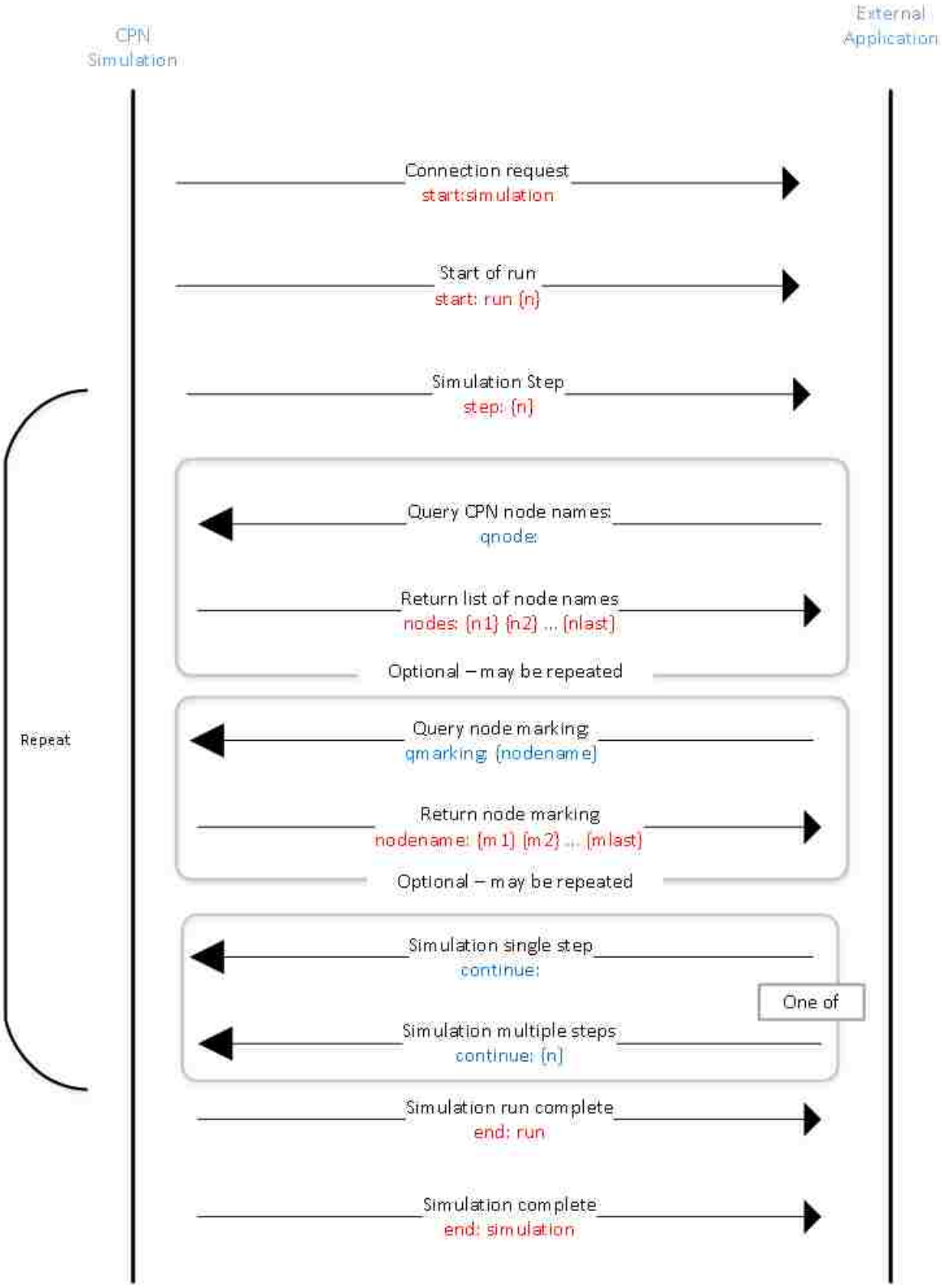

Figure 3.8. External Application Message Flow

4. APPLICATION TO CPS

Marashi has researched interdependencies between the cyber and physical components of a CPS [23]. This work explores interdependencies via case studies using IEEE-14 and IEEE-57 bus systems [29]. This work focuses on the IEEE-57 bus system. The IEEE-57 bus test case is a simple approximation of the America Electric Power system deployed in the Midwest U.S. in the early 1960s. There are 57 buses, 7 generators and 42 loads. An IEEE-57 model augmented with cyber control components is used in this work as the basis for tracking the flow of corrupted data. The focus of this work is on the impact to the critical components identified by Marashi. In this model, cyber control components; phasor measurement units (PMUs) and flexible AC transmission system (FACTS) devices and a central control unit are overlaid over the physical IEEE-57 bus system. The topology of the cyber components as defined by Marashi [Figure 4.3, page 37] is used as the basis for the colored Petri net model used in this case study.

A stepwise CPN simulation is then executed. The current state information of each place is logged, providing step-by-step detail of the CPN. This data becomes the input to the next step, analysis.

Analysis is performed comparing captured state data against predefined thresholds for total amount of permissible invalid data and number of consecutive steps where missing and corrupt data exceed predefined limits. Individual transmission line criticality values from previous simulations are used (Table 4.1).

Marashi's simulation was conducted over 25 steps. Simulations of the overlaid cyber topology of 25,000 steps each are executed, corresponding to 1,000 CPN steps per power systems step. A configurable threshold (3% used in this model) was established as the allowable percentage of corrupt/missing data for the central processing and PMU nodes. Data collected capture the step number where database corruption exceeds the

Table 4.1. Single Line Criticality (with FACTS)

Line	**Criticality**
l_{6-7} (F_3)	0.123
l_{1-16} (F_4)	0.066
l_{1-17} (F_5)	0.060
l_{6-8} (F_1)	0.060
l_{7-8} (F_2)	0.020
l_{54-55} (F_7)	0.020
l_{12-17} (F_6)	0.000

permitted threshold and number of steps duration. Consecutive over-threshold sequences ranged from a minimum of 1 step to maximum of 115 consecutive steps. Corruption data for each run and for each node was collected. Figure 4.4 shows an example of data from cyber component F_6 plotted per step for a single run. A trend-line showing the specified allowable corrupted data limit is included. Data point below the threshold are colored blue and green, those above the threshold orange and red. Table 4.2 shows database states of a 25,000 step simulation. Figure 4.1 shows this data graphically.

Table 4.2. CPN Databases: Percent Missing or Corrupt

	Run									
Node	1	2	3	4	5	6	7	8	9	10
l_{1-16} (F_4)	1.03%	0.91%	0.96%	0.96%	1.29%	1.12%	1.08%	1.38%	0.93%	1.21%
l_{1-17} (F_5)	1.05%	1.05%	1.20%	1.22%	1.23%	1.06%	0.69%	0.81%	1.07%	1.08%
l_{12-17} (F_6)	1.06%	1.25%	0.91%	0.95%	1.30%	1.22%	0.97%	1.03%	1.47%	1.05%
l_{54-55} (F_7)	1.31%	0.93%	1.04%	1.36%	1.13%	0.82%	0.90%	1.52%	1.10%	1.11%
l_{6-7} (F_3)	1.40%	0.88%	0.89%	0.86%	1.03%	0.91%	0.94%	0.94%	1.43%	0.98%
l_{6-8} (F_1)	0.90%	1.36%	1.33%	0.96%	1.19%	1.19%	1.43%	0.82%	1.22%	0.90%
l_{7-8} (F_2)	1.11%	1.27%	1.06%	1.19%	1.25%	1.09%	1.20%	1.04%	1.19%	0.92%
$dbProcessing$	2.10%	2.10%	2.33%	1.94%	2.07%	2.18%	2.00%	1.75%	2.16%	2.30%

To assess criticality of overlaid data processing elements, aggregated measures are calculated combining criticality of the underlying physical components with data corruption percentages from the data processing elements from the CPN model.

Physical aggregated criticality and fragility metrics are calculated for each PMU as:

Aggregate PMU Criticality

$$\forall p \in P : P = \text{PMU devices} \tag{4.1}$$

$$\forall l \in L : L = \text{lines attach to bus L} \tag{4.2}$$

$$c(l_{i-j}) = \text{calculated criticality of line } l_{i-j} \tag{4.3}$$

$$C_p = \sum_{l_{i-j} \in L} c(l_{i-j}) \tag{4.4}$$

Aggregate measures and information from the CPN model, specifically database state, are then combined to calculate a figure-of-merit that quantifies the impact corrupt and missing data have to the system and which cyber components are most at risk of impacting system service delivery when affected by corrupt or missing data.

Using the criticality and average database state measures the impact metric can be calculated as:

Impact

$$\text{Impact: } I_p = [\text{Average Percentage Corrupt Data}] * C_p * 100 \tag{4.5}$$

Calculated values for FACTS devices are shown in Table 4.3. Comparison of criticality measures between the Single Line Criticality in Table 4.1 against the Aggregate Criticality and Impact in Table 4.3 show different orders. The Criticality value of 0.000 listed in the table for FACTS device F_6 (line l_{12-17}) reflects improvements to service delivery from the addition of an active power flow device as compared to a physical only IEEE-57 bus system.

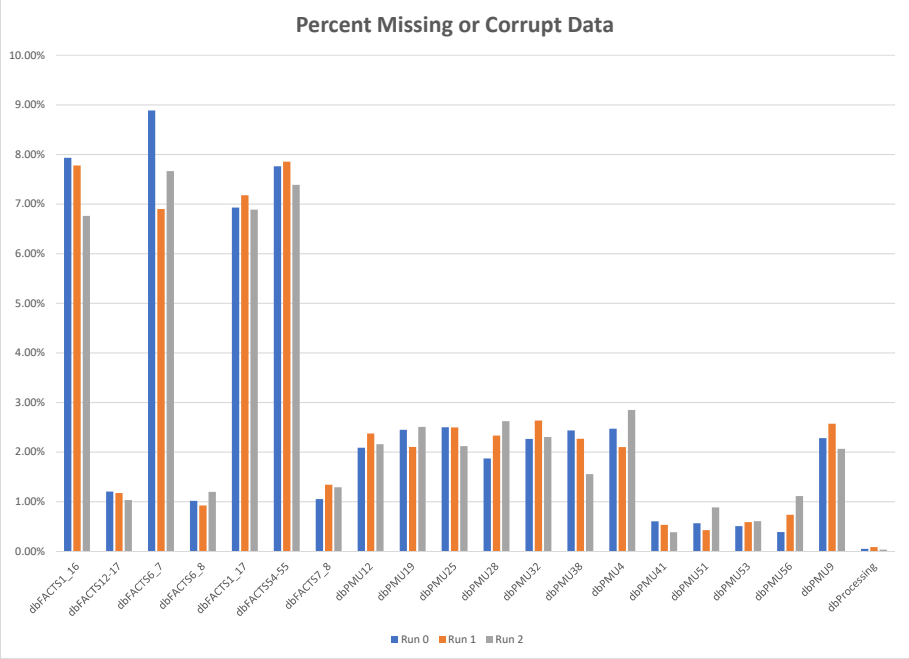

Figure 4.1. Average Corrupted or Missing Data

While single device criticality analysis provides one measure, the impact metric showing the effect of missing and corrupt data shows a different order. The Impact measure provides an additional mechanism for systems designers to use as they look at improving systems reliability.

Table 4.3. FACTS Line Aggregate Criticality and Impact

Data Processing Node	**Extent of Data Corruption (Expected)**	**Missing and Corrupt Data Criticality**	**Data Corruption Impact (Expected)**
l_{6-8} (F_1)	1.13%	0.80	0.91
l_{7-8} (F_2)	1.13%	0.69	0.78
l_{6-7} (F_3)	1.02%	0.66	0.67
l_{1-16} (F_4)	1.09%	0.63	0.68
l_{1-17} (F_5)	1.05%	0.57	0.59
l_{12-17} (F_6)	1.12%	0.28	0.31
l_{54-55} (F_7)	1.12%	0.06	0.07

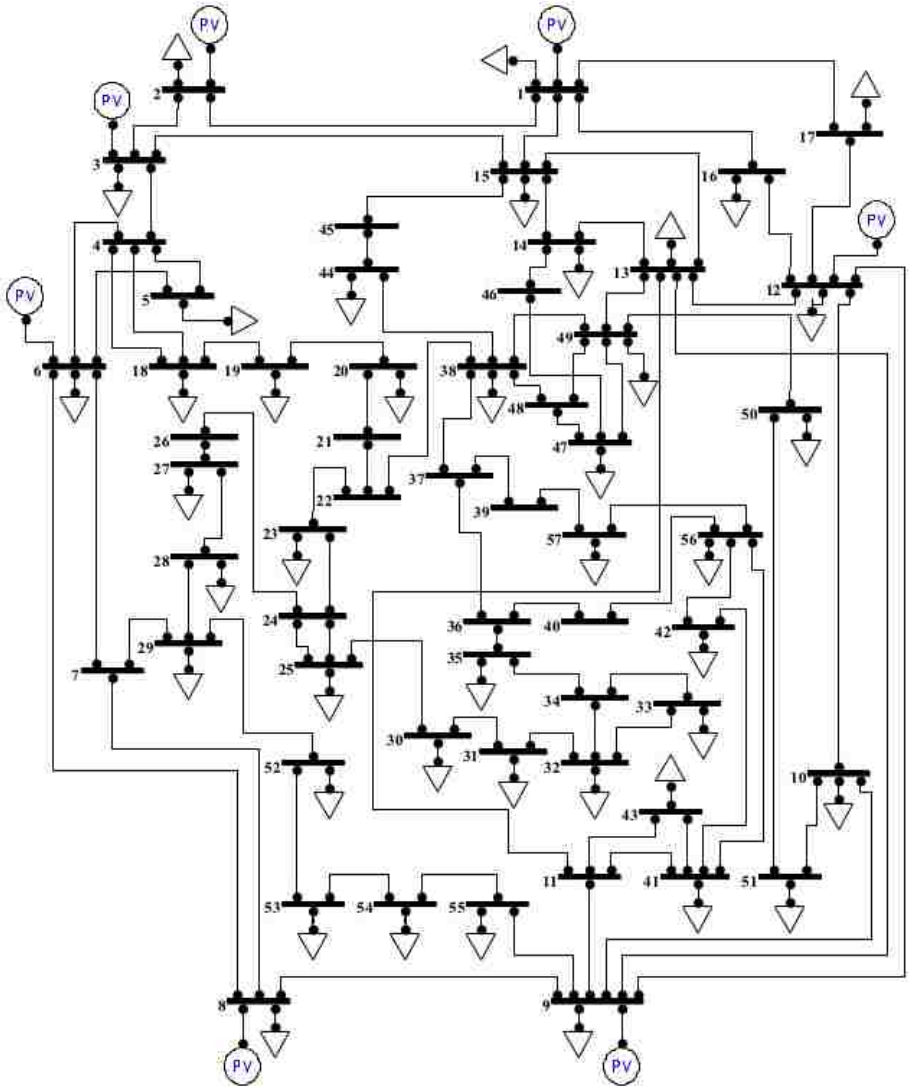

Figure 4.2. IEEE-57 Bus Smart Grid

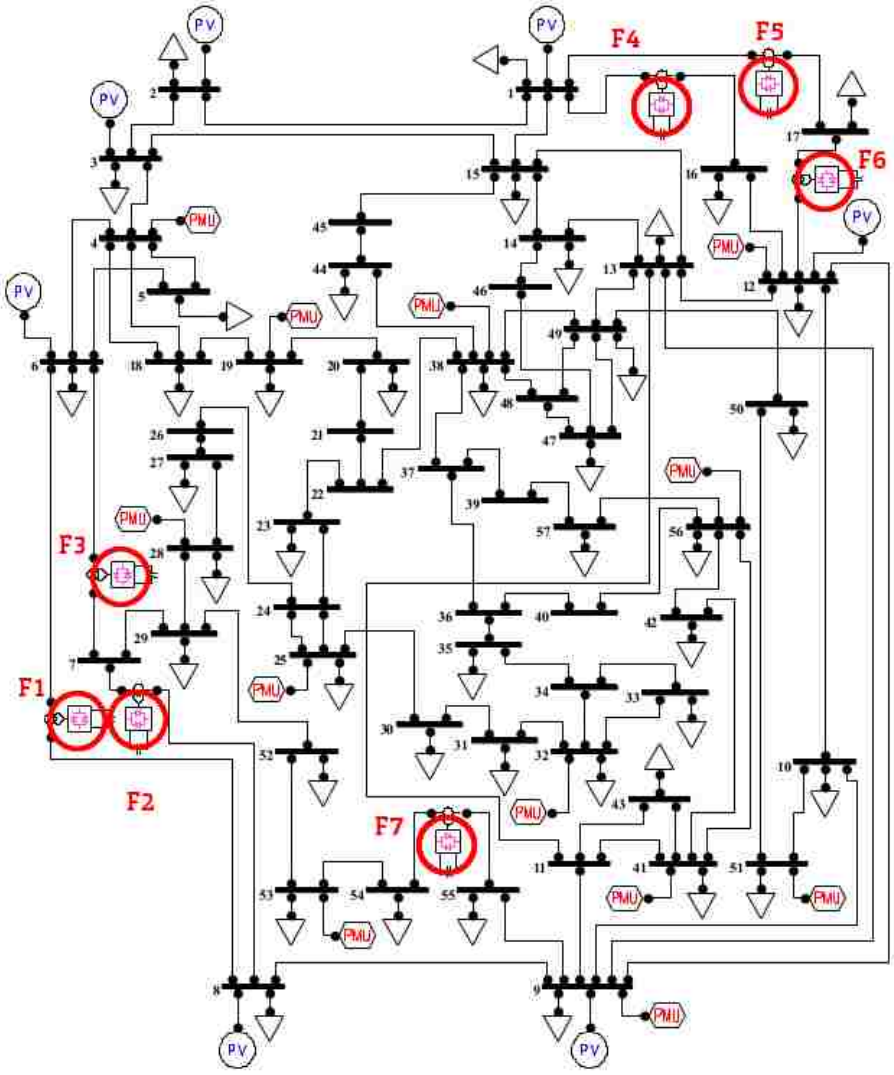

Figure 4.3. IEEE-57 Bus Smart Grid with Cyber Control Components

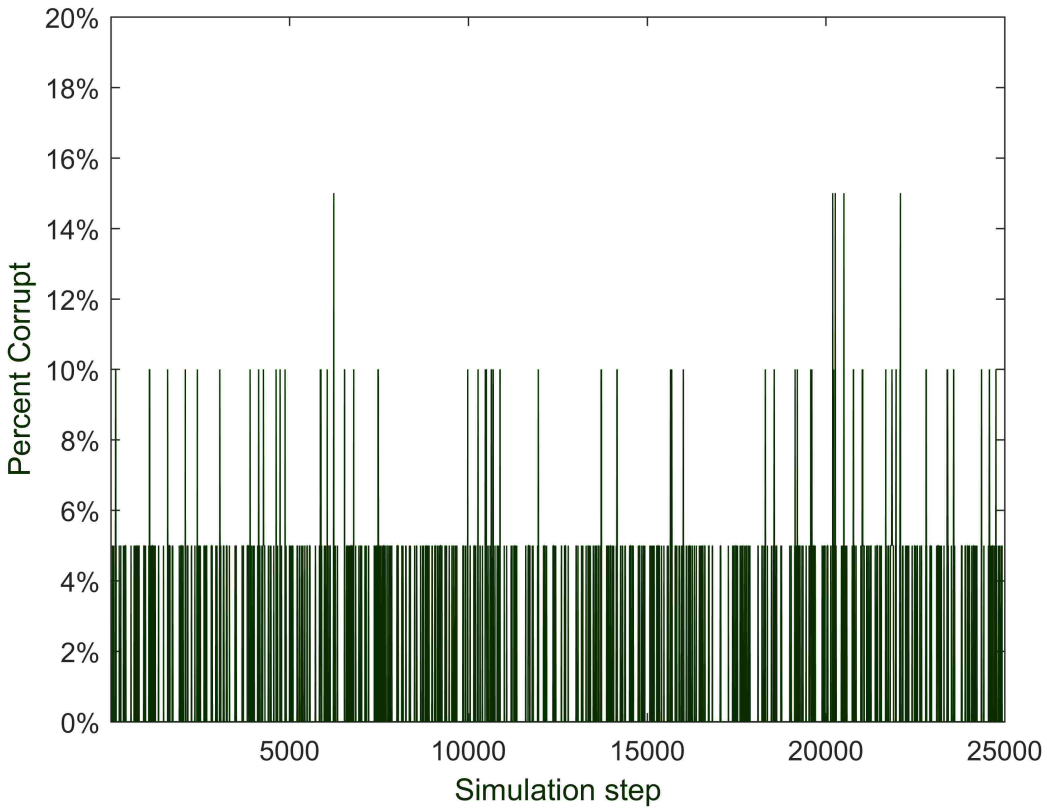

Figure 4.4. Percentage Corrupt or Missing Data

5. CONCLUSIONS AND FUTURE WORK

This thesis proposes a method for predicting the impact of data corruption at each data processing node of a CPS, based on:

- The extent of data corruption expected at the node
- The consequences of the node's failure in terms of service interruptions, i.e., its criticality

A second contribution of this work is a performance improvement to a colored petri net simulation. As this model is applied to more complex cyber physical system topologies, the improved performance enables analysis of larger models. Replacement of transition guard design with Python language functions will allow future work to include more complex failure probability logic. For example, corruption probabilities can be made dynamic, using distributions such as Exponential, Weibull, Poisson, or even modeler-defined as applicable, to meet specific needs.

A third contribution is enabling of integration of the data corruption modeling with external tools. Interfaces have been developed allowing communication to external tools at each modeling step. With the communication capability, external tools can take actions based on current CPN marking, affect CPN markings, or step-by-step marking analysis. The communication mechanism utilizes standard TCP/IP socket techniques and, as such, enables connection to a wide variety of external tools.

An example application to an electrical distribution system was demonstrated. It was demonstrated that combining importance analysis results with the colored Petri net simulation results can provide predictive assistance to evaluate impact of data processing element failure caused by corrupted or missing data. These predictive values can be used to identify data processing elements where data corruption issues have the highest impact on

system service delivery and where investments are required to improve systems reliability. The proposed method aids in making determination of system, component or network modifications necessary to meet service level requirements. As important as identification of where investments are needed, knowledge gained from this method can also be used to identify areas where hardening investments are of limited value. Identification of areas where minimal service delivery impact is realized by hardening against failure allows for resources to be applied to more critical areas of the cyber-physical system, resulting in both improved service delivery and lower overall cost.

With the enabling contribution of this work it is anticipated that future work will extend this research in the following areas:

- The model as described, is a template of idealized physical devices. Future work can include developing models that more closely mirror actual physical devices characteristics - the amount of missing or corrupt data than can be tolerated, both an absolute percentage of missing or corrupt data at any one time and a threshold over time (consecutive steps).

- Variations of the internal node database size can be explored to determine impact of corrupted data and what tolerance levels provide acceptable service delivery.

- The model described captures raw state space data. Petri net state space analysis tools and methods can be used against this data to quantify, describe and better understand missing and corrupt data propagation patterns. Additional insight into model behavior can be realized by performing reachability analysis of the captured state data.

APPENDIX

EXAMPLE TRANSITION PROBABILITY FUNCTIONS

The following are examples of modeler written Python transition functions as described in Section 3.

```
"""
Probability distribution functions
"""
__author__ = "Erik Burgdorf, edb4k4@mst.edu"

from random import random, randint

def processingFunction(x, pProcCorruption, pProcMiss, db_size):
  """
  Data corruption processing function

    Inputs:

    x: token database
    pProcCorruption: probability of processing corruption
    pProcMiss: probability processing creates missing data
    db_size: number of tokens in database

    Returns:

    database with updated entries
  """

  dbcontent = x.items()
  slot = randint(0, len(dbcontent)-1)
  p_proc = random()

  if p_proc <= pProcCorruption:        # proc_corr
    dbcontent[slot] = 1            #    CORRUPT
  elif p_proc > (1.0-pProcMiss):         # proc_miss
    dbcontent[slot] = 2            #    MISS
  else:                     #
    dbcontent[slot] = 0            #    VALID
```

```
#
# trim db to appropriate number of entries
#

  while len(dbcontent) > db_size:
    del dbcontent[randint(0, len(dbcontent)-1)]

  return dbcontent

def sensorFunction(x, pSensorCorruption, pSensorMiss):
  """
  Sensor function

    Inputs:

    x: data token
    pSensorCorruption: probability sensor corrupts token
    pSensorMiss: probability sensor creates missing data

    Returns:

    data token
  """

  p = random()
  if p <= pSensorCorruption:           # sens_corr
    r = 1                    #    CORRUPT
  elif (1.0-pSensorMiss) < p:          # sens_miss
    r = 2                    #    MISS
  else:                      #
    r = 0                    #    VALID
  return r

def detectFunction(x, pTruePositive, pFalsePositive):
  """
  Detection function

    Inputs:

    x: data token
    pTruePositive: probability corruption correctly detected
    pFalsePositive: probability corruption incorrectly detected

    Returns:

    data token
    flagged indication
```

```
  """

  r = x
  flagged = False

# detect

  p = random()                  # p(detection)

  if x == 0 and 0 <= p < pFalsePositive:
    r = 0                   # False positive
    flagged = True
  elif x == 1 and 0 <= p < pTruePositive:
    r = 1                   # True positive
    flagged = True
  elif x == 2:
    r = 2                   # Missing
    flagged = True
  elif x == 0 and pFalsePositive <= p < 1.0:
    r = 0                   # True negative
    flagged = False
  elif x == 1 and pTruePositive <= p < 1.0:
    r = 1                   # False negative
    flagged = False

  return (r, flagged)

# mitigate

def mitigateFunction(x, flagged, pFix, pDrop):
  """
  Mitigation function

    Inputs:

    x: data token
    flagged: indication if token was flagged as corrupt
    pFix: probability corrupt token is corrected
    pDrop: probability token is dropped (made missing)

    Returns:

    data token
    flagged indication
  """

  r = x
```

```
  if flagged is True:
    p = random()               # p(mitigation)

    if r == 1 or r == 2:           # bad and missing
      if 0 <= p < pFix:
        r = 0              #     mitigate
      elif pFix <= p < pDrop:
        r = 1              #     unable to mitigate
      else:
        r = 2              #     drop
    elif r == 0:               # good
      if 0 <= p <= pDrop:
        r = 1              #     false mitigation
      elif pDrop <= p < 1:
        r = 2              #     false drop

  return r

def communicationFunction(x, pCommCorruption, pCommMiss):
  """

  Communication function

    Inputs:

    x: data token
    pCommCorruption: probability communication corrupts token
    pCommMiss: probability communication creates missing data

    Returns:

    data token
  """

  p = random()
  if 0.0 <= p <= pCommCorruption:        # comm_corr
    r = 1                  #    CORRUPT
  elif (1.0-pCommMiss) <= p < 1.0:       # comm_miss
    r = 2                  #    MISS
  else:                    #
    r = 0                  #    VALID
  return r
```

REFERENCES

[1] J. H. Van Beek, A.-C. Hauschild, H. Hettling, and T. W. Binsl, "Robust modelling, measurement and analysis of human and animal metabolic systems," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1895, pp. 1971–1992, 2009.

[2] Y.-S. Huang, Y.-S. Weng, and M. Zhou, "Modular design of urban traffic-light control systems based on synchronized timed petri nets," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 530–539, 2014.

[3] Schiaparelli Inquiry Board - European Space Agency, "ExoMars 2016 - Schiaparelli Anomaly Inquiry," tech. rep., 2017.

[4] Bureau d'Enquêtes et d'Analyses pour la sêcuritê de l'aviation civile, "Final report on the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro–Paris," tech. rep., 2012.

[5] "Report of the Enquiry Committee on Grid Disturbance in Northern Region on 30th July 2012 and in Northern, Eastern & North-eastern Region on 31st July, 2012.," tech. rep.

[6] J. Voas, "Error propagation analysis for COTS systems," *Computing Control Engineering Journal*, vol. 8, pp. 269–272, Dec 1997.

[7] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 400–412, Dec 2007.

[8] D. Behrens, *Error isolation in distributed systems*. PhD thesis, Technische Universitat Dresden, 2015.

[9] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, Jan 2004.

[10] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 193–204, ACM, 2009.

[11] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design," in *ACM SIGPLAN Notices*, vol. 47, pp. 111–122, ACM, 2012.

[12] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Ieee Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[13] S. Borkar *et al.*, "Microarchitecture and design challenges for gigascale integration," in *MICRO*, vol. 37, pp. 3–3, 2004.

[14] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. G. Dhoot, A. R. Kumar, A. Agiwal, *et al.*, "Mesa: Geo-replicated, near real-time, scalable data warehousing," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1259–1270, 2014.

[15] W. Stallings and M. P. Tahiliani, *Cryptography and network security: principles and practice*, vol. 6. Pearson London, 2014.

[16] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 300–321, Springer, 2016.

[17] M. A. Friedman, P. Y. Tran, and P. I. Goddard, *Reliability of software intensive systems*. William Andrew, 1995.

[18] M. Beccuti, S. Chiaradonna, F. Di Giandomenico, S. Donatelli, G. Dondossola, and G. Franceschinis, "Quantification of dependencies between electrical and information infrastructures," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 1, pp. 14–27, 2012.

[19] M. Malhotra and K. S. Trivedi, "Dependability modeling using Petri-nets," *IEEE Transactions on Reliability*, vol. 44, pp. 428–440, Sept. 1995.

[20] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, "Petri Net Modeling of Cyber-Physical Attacks on Smart Grid," *IEEE Transactions on Smart Grid*, vol. 2, pp. 741–749, Dec. 2011.

[21] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *Machine Learning in Cyber Trust*, pp. 3–13, Springer, 2009.

[22] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin, "Catastrophic cascade of failures in interdependent networks," *Nature; London*, vol. 464, no. 7291, pp. 1025–1028, 2010.

[23] K. Marashi, S. S. Sarvestani, and A. R. Hurson, "Quantification and analysis of interdependency in cyber-physical systems," in *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*, pp. 149–154, IEEE, 2016.

[24] M. Woodard, S. Sedigh Sarvestani, and A. R. Hurson *Reliability Engineering & Safety Systems*, 2017 Submitted. Reliability Engineering & Safety Systems.

[25] L. M. Kristensen and K. Jensen, *Specification and validation of an edge router discovery protocol for mobile ad hoc networks*, pp. 248–269. Springer, 2004.

[26] J. Huang, Y. Zhu, B. Cheng, C. Lin, and J. Chen, "A Petri net based approach for supporting traceability in cyber-physical manufacturing systems," *Sensors*, vol. 16, p. 382, Mar 2016.

[27] M. K. Molloy, "Discrete time stochastic Petri nets," *IEEE Transactions on Software Engineering*, no. 4, pp. 417–423, 1985.

[28] M. Woodard, K. Marashi, and S. Sedigh Sarvestani, "Survivability evaluation and importance analysis for complex networked systems," *IEEE Transactions on Network Science and Engineering*, 2017.

[29] R. Christie, "Power Systems Test Case Archive, University of Washington." `https://www2.ee.washington.edu/research/pstca`, 2000.

VITA

Erik David Burgdorf was born in Ohio and has lived and worked in Texas, North Carolina, and Missouri. He attended the University of Missouri - Rolla (now the Missouri University of Science and Technology) from 1975 through 1980, earning a Bachelor of Science in Computer Science in May 1980. Following graduation he worked in industry, contributing to graphics terminal design, embedded systems development, real-time operating system design, data and voice switching systems, and large scale data analysis software. After 26 years in the industry, he chose to apply his skills in support of a non-profit charitable foundation. He spent 10 years as Director, managing the foundation corpus and overseeing grant processes and beneficiary relationships. Upon completion of the foundation work he made the decision to return to his alma mater and complete a graduate degree, receiving a Master of Science degree in Computer Engineering in December 2017.