MODELS OF LEADER ELECTIONS AND THEIR APPLICATIONS

by

STEPHEN CURTIS JACKSON

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2016

Approved

Dr. Bruce McMillin, Advisor
Dr. Ali Hurson
Dr. Wei Jiang
Dr. Sriram Chellappan
Dr. Sahra Sedigh Sarvestani

ABSTRACT

New research about cyber-physical systems is rapidly changing the way we think about critical infrastructures like the power grid. Changing requirements for the generation, storage, and availability of power are all driving the development of the smart-grid. Many smart-grid projects disperse power generation across a wide area and control devices with a distributed system. However, in a distributed system, the state of processes in that system is hard to determine. By using information flow security models, we reason about a process's beliefs of the system state in a distributed system. Information flow analysis aided in the creation of Markov models for the expected behavior of a cyber controller for a smart-grid system using a communication network with omission faults. The models were used as part of an analysis of the distributed system behavior when there are communication faults. With insight gained from these models, existing congestion management techniques were extended to create a feedback mechanism, allowing the cyber-physical system to better react to issues in the communication network.

ACKNOWLEDGMENTS

It is hard to believe I've spent nearly a decade at Missouri S&T. I am eternally grateful to everyone at this university who helped me along the way. Bruce McMillin got me started down this long path of research when he hired me to work with David Cape and has provided me with so many opportunities. I have had so many wonderful experiences with everyone I've worked with in the Computer Science department.

Special thanks to Ali Hurson, Wei Jiang, Sahra Sedigh Sarvestani, and Sriram Chellappan for their service on my Ph.D. committee. I'm very grateful to every instructor I've had at S&T who has inspired me to explore and learn. Thanks to Dawn Davis, Christina Barton, and Rhonda Grayson for their years of administrative support. Thanks to all my lab mates: Li Feng, Thomas Roth, Gerry Howser, and everyone else along the way for making our lab such a great place to work.

My research was supported by the two excellent programs: the Future Renewable Electric Energy Delivery and Management Center, a National Science Foundation-supported Engineering Research Center under grant NSF EEC-081212, and the United States Department of Education GAANN program.

Special thanks to my family: Sue, Kent, Laura, and everyone else who encouraged me and supported me through this long journey. I also want to give thanks to all of my friends for their support: Emily, Drew, Doug, Tony, Michaela, Liz, Robert, Remington, Dwight and everyone else at KMNR; you've made this long stay at S&T a lot more comfortable. I love you all.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

LIST OF ACRONYMS

FREEDM Future Renewable Electric Energy Delivery and Management

DGI Distributed Grid Intelligence

CPS Cyber-physical systems

AYC Are You Coordinator

AYT Are You There

RED Random Early Detection

NS-3 Network Simulator 3

ECN Explicit Congestion Notification

VANET Vehicular Ad Hoc Networks

EWMA Exponentially Weighted Moving Average

CDF Cumulative Distribution Function

BIT Belief, Information transfer, Trust

HMM Hidden Markov Model

FIFO First In First Out

DTMC Discrete Time Markov Chain

CTMC Continuous Time Markov Chain

pdf probability distribution function

DF degrees of freedom

CV critical value

IGT in group time

AGS average group size

AP atomic proposition

SCADA Supervisory Control And Data Acquisition

IoT Internet of Things

M2M Machine to Machine

DNP3 Distributed Network Protocol 3

DoS Denial of Service

wff Well-formed formula

SRC Sequenced Reliable Connection

SUC Sequenced Unreliable Connection

MSDND Multiple Security Domain Model Non-deducibility

NOMENCLATURE

A	Availability of a component.
B_i	The modal belief operator.
D_i	The domain of an agent or process i.
$E[X]$	Expected value of a random variable.
$I_{i,j}$	The modal information transfer operator.
K	A Kripke model.
P	Transition probability matrix of a discrete time Markov chain.
R	Set of relations between worlds in a Kripke model or frame.
$R(t)$	Reliability of a component.
S	Set of boolean state variables for a Kripke model.
T	Test chain. A Markov chain sampled from a system.
$T_{i,j}$	The modal trust operator.
V	A valuation function.
W	Set of worlds for a Kripke model or frame.
X	A random variable, or the set of states for a Markov chain
Y	A random variable or the set of observable states for a hidden Markov model
$\Box\varphi$	The modal "it is necessary that" operator.

$\Diamond\varphi$ The modal "it is possible that" operator.

$\mathbb{V}$ Set of valuation functions in a Kripke model.

φ, γ, ψ Well-formed formulas.

1. INTRODUCTION

The design of stochastic models of distributed systems has a long history as a challenging area of interest. Models of distributed systems have to consider many factors, such as various types of failure the system could experience, a lack of tightly synchronized execution, and a large complex state space when there are many agents[30][8]. However, the concept of distributed systems plays a central role in how critical infrastructures will operate in the future. These critical infrastructures are physical networks whose operation are so vital that if those networks failed to operate correctly it would be highly detrimental to the population that relies on those systems. Cyber-physical systems (CPS) are the integration of computational systems with physical networks. Computational systems already play a major role in most critical infrastructures, and as demand for security features, such as accessibility, increases, distributed systems are a favorable choice for the computational needs for these systems[68].

The Future Renewable Electric Energy Delivery and Management (FREEDM) center[53], envisions a future power grid with widely distributed renewable power generation and storage closely coupled with a distributed system that facilitates the dispatch of power across those areas. Other systems like Vehicular Ad Hoc Networks (VANET)[45][57][31] and air traffic control systems[61][6] also propose similar control systems where many computers must cooperate to ensure both smooth operation and the safety of the people using those systems. As a consequence, ensuring that the computer systems that control those infrastructures behave correctly during fault conditions is critical, especially when those computer systems rely on their interaction with other computers to operate.

A robust CPS should be able to survive and adapt to communication network outages in both the physical and cyber domains. When an outage occurs, the physical or cyber components must take corrective action to allow the rest of the system to continue operating normally. Additionally, processes may need to react to the state change of some other process. Managing and detecting when other processes have failed is commonly handled by a leader election algorithm and failure detector.

In a smart-grid system, misbehavior during fault conditions could lead to critical failures such as a blackout or voltage collapse. In a VANET or air traffic control system, vehicles could collide, injuring passengers or destroying property. Additionally, because these systems are a part of critical infrastructure, protecting them from malicious entities is an important consideration.

We were motivated by observations on the effects of lost messages on the group management module of the Distributed Grid Intelligence (DGI) used by the FREEDM smart-grid project. The original observations confirmed the need to explore more well-defined models for the behaviors of CPS in order for them to better serve the people who use them.

We present a framework for reasoning about inferable state in the context of a distributed system. To do this, we exploit existing work in the field of information flow security, which has been used to reason how attacks like Stuxnet can manipulate operators' beliefs while disrupting a system[25]. In particular, approaches in information flow security reason how the operator in a Stuxnet attack has no avenue to verify the reports from a compromised computing device. Using existing modal logic frameworks and information flow security models[36][25][37], one can formally reason where information, not normally known to a domain, can be inferred.

With the correct information flows, an agent in a distributed system can infer the state of other agents in the system. With this information, an agent can construct

a reasonable model of the system to determine if the current behavior could lead to an undesirable situation with either the cyber or physical network.

Using the framework, we present a leader election algorithm which can be modeled with a Markov chain for a known omission fault[19] rate. The presented algorithm maintains the Markov property for the observations of the leader despite omission faults. Our approach to considering how a distributed system interacts during a fault condition allows for the creation of new techniques for managing a fault scenario in cyber-physical systems. In the context of FREEDM, these models produce expectations of how much time the DGI will be able to spend coordinating and doing useful work. With this information, the behavior of the control system for the physical devices can be adjusted to prevent faults.

We also propose using existing schemes to detect communication network congestion and inform processes in a CPS of impending congestion. Processes act on congestion information to change their behavior in anticipation of message delays or loss. Using an alternative behavior allows them to harden themselves against the congestion, and allows them to continue operating as normally as possible during the congestion. The technique involves changing the behavior of both the leader election[29] and physical device management algorithm during congestion.

To detect and inform agents of that congestion, we extend existing networking concepts of Random Early Detection (RED), Explicit Congestion Notification (ECN)[56], and ICMP source quench[9]. When a network device detects congestion, it notifies processes of the congestion, and they should react appropriately. We demonstrate an implementation of the FREEDM DGI in a Network Simulator 3 (NS-3) simulation environment[18] with our congestion detection feature. The DGI operates normally until the simulation introduces a traffic flow, congesting the network devices in the simulation. After congestion has been identified by the RED queuing algorithm, the DGI are informed. When the congestion notifications are

introduced, the DGI maintains configurations which they would normally be unable to maintain during congestion. Additionally, we show a greater amount of work can be done without the work causing unstable power settings to be applied.

2. BACKGROUND

2.1. DISTRIBUTED SYSTEMS

Distributed systems are a computing paradigm characterized by independence of computational units and no universal clock. Components in a distributed system may not directly share computational resources or memory. Instead, computers in a distributed system typically interact through a message-passing interface.

As a result, distributed computing is a challenging area of research. In a distributed system, because processes do not share a universal clock, the ordering of messages and events needed to be carefully considered to ensure correct operation of the system. Additionally, when individual components fail, determining which components failed and how they affected the system as a whole is difficult in a distributed system. Different types of failures can cause different kinds of information to be withheld or changed, disrupting those processes.

2.2. EXECUTION AND COMMUNICATION MODELS

We consider a distributed system where no processes in the system share an address space. All processes must use a message-passing interface to communicate with other processes to exchange information. As a result of the complexities of distributed systems, various execution models have been developed to define how the execution of a distributed system proceeds. Different execution models affect how easy it is to reason about the system's execution, the types of algorithms the system can execute, and how complicated they are to implement.

2.2.1. Communication Channels. We describe the avenues of communication between processes as channels. A channel is classified as reliable if it meets three axioms[30]:

Axiom 1. *Every message sent by a sender is received by a receiver and every received message was sent by a sender in the system.*

Axiom 2. *Every message has an arbitrary but not infinite propagation delay.*

Axiom 3. *Every channel is a First In First Out (FIFO) channel. If process P sends messages x and y to Q (in that order) then Q receives the messages in order (x then y).*

Reliable channels are often referred to as synchronous channels. In our system, however, channels are not assumed to be perfectly reliable. Instead, we respect Axiom 3, partially fulfill Axiom 1, and disregard Axiom 2. Therefore, we assume the following about communication channels in our systems (replacing Axiom 1):

Axiom 4. *Every message received by a receiver was sent by a sender in the system.*

Without the constrained propagation delay from Axiom 2, this type of communication channel is typically referred to as an asynchronous channel.

The communication model can be synchronous or asynchronous. In the synchronous communication model, processes can only send a message when the receiving process is ready to receive it. Algorithmically, sending in a synchronous model is usually considered a "blocking" operation, meaning, once a process tries to send a message, it cannot proceed until the message is received. In this work, communication is asynchronous: a process does not wait for the successful delivery of a message, typically referred to as a non-blocking send.

2.2.2. Clocks. Clocks are considered synchronized if every clock in the system reads the same time. Since it is impossible for independent clocks to tick at the

same rate, weak synchronization is used to describe when clocks in the system have an upper bound on drift rate from each other.

2.2.3. Execution. In a system with synchronous processes, processes execute in lockstep. At each step, a process executes its next available action. Synchronous execution requires the strong organization of the processes executing the algorithm. The systems presented rely on partially synchronous execution by processes. In the partially synchronous model, execution proceeds in rounds or phases. The start of each round or phase is synchronized between processes, using a synchronized clock.

2.3. FAULTS, FAILURES, AND ERRORS

Processes can encounter incorrect behavior or issues during execution. Errors, faults, and failures describe the severity and consequence of the issue.

Definition 1. *An error is a difference between what is considered "correct" output for a given component, and the actual output: an incorrect result.*

Definition 2. *A fault is the manifestation of an error in software or an incident where an incorrect step or data definition is performed in a computer program.*

Definition 3. *A failure is the inability of a component or system to perform its required function or within its specified limits.*

2.3.1. Crash or Fail-Stop Failure. A crash failure, or its more generalized form, a fail-stop failure, describes a failure in which a process stops executing. In general, it is considered to be an irreversible failure, since a process typically does not resume from a crashed state. There is a special category of crash failures, called napping failures, where a process will appear to have crashed for a finite amount of time before resuming normal operation. Crash failures are impossible to detect

with absolute certainty in an asynchronous system. Processes can be suspected of failure by other processes through the use of challenge/response messages or heartbeat messages that allow a process to prove that it has not crashed yet. A system that can handle a crash failure is implied to be able to handle a fail-stop failure.[30]

The fail-stop failure has three properties [30]:

1. When a failure occurs the program execution is stopped.
2. A process can detect when another fail-stop process has failed.
3. Volatile storage is lost when the fail-stop process is stopped.

2.3.2. Failure Detectors. Failure detectors[13] (sometimes referred to as unreliable failure detectors) are a special class of processes in a distributed system used to detect other failed processes. Distributed systems use failure detection to identify failed processes for leader election routines. Because it isn't possible to directly detect a failed process in an asynchronous system, there has been a wide breadth of work related to different classifications of failure detectors with different properties. Some of the properties include[13]:

- Strong Completeness - Every faulty process is eventually suspected by every other working process.
- Weak Completeness - Every faulty process is eventually suspected by some other working process.
- Strong Accuracy - No process is suspected before it fails.
- Weak Accuracy - There exists some process that is never suspected of failure.
- Eventual Strong Accuracy - There is an initial period where strong accuracy is not kept. Eventually, working processes are identified as such, and are not suspected unless they actually fail.

- Eventual Weak Accuracy - There is an initial period where weak accuracy is not kept. Eventually, working processes are identified as such, and there is some process that is never suspected of failing again.

One class of failure detectors, Omega class Failure detectors, are particularly interesting because of [32]. An eventual weak failure (weak completeness and eventual weak accuracy) detector is the weakest detector which can still solve consensus. It is denoted several ways in various works including $\diamond\mathcal{W}$ [13], $\mathcal{W}$ [12] [14] and Ω (Omega) [32].

2.3.3. Omission Fault. An omission fault[19] occurs when a message is never received by the intended recipient. Omission faults can occur when the communication medium is unavailable or when the latency of message exceeds a timeout for its expected delivery. Protocols like TCP do not allow omission failures: a packet is resent until it is acknowledged by the receiver. If the acknowledgment never comes, the connection is closed. As a contrast, UDP assumes that any datagram could be lost, and it is the responsibility of the programmer to handle missing datagrams appropriately. An omission failure can have the same observable effects as a napping failure in some situations.[30]

2.3.4. Byzantine Fault. A Byzantine fault causes processes in the distributed system to send information that is incorrect or misleading to other processes. Constraints for detecting processes exhibiting Byzantine behavior are a famous result in distributed systems.[46]

2.4. PROBABILITY

Several concepts are useful for reasoning about the stochastic properties of a distributed system. The expected value represents the long-term average output of a probability distribution.

$$E[X] = x_1 p_1 + x_2 p_2 + ... + x_k p_k \tag{2.1}$$

Conceptually, the expected value is the weighted average of the outcomes of a stochastic system.

Definition 4. *Availability is the probability a system or component is operational and accessible when required for use, denoted A.*

$$A = \frac{E[uptime]}{E[uptime] + E[downtime]} \tag{2.2}$$

Definition 5. *Reliability is the ability of a system or component, in specified conditions, to be able to perform its required functions for a specified period.*

Let R(t) be the probability a system or component function up until at least time t:

$$R(t) = \Pr(T > t) = \int_t^\infty f(x)dx \tag{2.3}$$

where $f(x)$ is the probability density function for the component's survival. Unfortunately, there is a clash of the term for reliability in distributed systems and reliability analysis. We try to constrain the discussion about reliability in the distributed systems sense to communication channels.

2.5. MARKOV MODELS

A Markov chain is a finite set of states $X = \{x_1, x_2, ..., x_n\}$ and probabilistic transitions between those states. States in a Markov chain are mutually exclusive.

When a system is some state x_i it has some probability of transitioning to some other state x_j at the next time-step. A Markov chain is a first order chain if the probability of transitioning from i to j does not depend on the history of transitions that lead to state i. First order chains are described as being memoryless because they have the Markov property. The Markov property formalizes the independence of the next state from the history of previous states. If we describe a Markov chain as a sequence of random variables $X_1, X_2, X_3, ...$, the Markov property states the value of X_{n+1} only depends X_n: [10]

$$\begin{aligned} &\Pr(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &\quad = \Pr(X_{n+1} = x \mid X_n = x_n). \end{aligned} \tag{2.4}$$

An ergodic Markov chain is a chain where it is possible, in some finite number of steps, to go from any state to any other state. A stationary, or time homogeneous, Markov chain is one where the transition probabilities do not change over time. In a stationary Markov chain, the nth visit to a state is indistinguishable from the $(n+1)$th visit to a state.

2.5.1. Discrete Time Markov Chain. A Discrete Time Markov Chain (DTMC) is one where the transitions between states happen at discrete time steps. A DTMC with m states can be represented by an $m \times m$ matrix. For simplicity when creating the model, matrices in this work are 1-indexed. In a matrix P, the value of P_{ij} represents the probability of the transition from x_i to x_j. The matrix is row stochastic, meaning the sum of each row in the matrix is equal to one:

$$\sum_{i=1}^{m} P_{ij} = 1. \tag{2.5}$$

A useful companion to the transition matrix is a state distribution vector. While the transition matrix describes how a system will transition between states, the state distribution vector describes the probability of observing a given state.

Definition 6. *A state distribution vector is an m-dimensional vector composed of the probability of observing each state in the system at a given instant.*

Let v be a state distribution vector:

$$v = [P_1 \quad P_2 \quad \dots \quad P_m]$$

where P_i corresponds to the probability of observing state x_i.

A DTMC is a suitable model for a memoryless random process with a finite number states which is observed at fixed time intervals. By utilizing a Markov chain, a variety of statistical analyses can be performed on the modeled system. For example, a Markov chain with the stationary and ergodic properties can be analyzed for its steady state probabilities. The steady state is a state distribution vector that describes the probability a random observation of a long-running process will observe some state x_i. The steady state probability distribution vector can by found via a system of equations: [10]

$$0 \leq \pi_j \leq 1.0 \tag{2.6}$$

$$\sum_{j=1}^{m} \pi_j = 1.0 \tag{2.7}$$

$$\pi_j = \sum_{i=1}^{m} \pi_i p_{ij}. \tag{2.8}$$

The computation of the steady state will be noted as $Steady()$. A Markov chain can also be used to predict what state a process will be in at some point in

the future. Given an initial state and a number of time-steps, a matrix operation will yield the likelihood of the process being in each state after the time interval has passed. The mean passage time, a measure of how many time-steps will pass before a process returns or arrives in some state, can also be calculated.

We model a leader election algorithm with a closed-form representation of the behavior of the algorithm. The closed-form representation is a profile Markov chain (noted as P) and will be validated against a chain generated from execution of the algorithm. The chain constructed from sampled data is known as a test chain (noted as T).

2.5.2. Continuous Time Markov Chain. Transitions in a Continuous Time Markov Chain (CTMC) depend on the amount of time spent in a given state. Let $X(s) = x_i$ indicate the model is in state x_i at time s. If the model is time homogeneous, then the probability of transitioning to state x_j only depends on the time spent in the current state (t).

$$\Pr\{X(s+t) = x_j | X(s) = x_i\} = \Pr\{X(t) = x_j | X(0) = x_i\} \tag{2.9}$$

Each transition has some expected value or holding time which describes the amount of time before a transition occurs. The probability distribution function (pdf) of the exponential distribution can be written as: [55]

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}. \tag{2.10}$$

As a result, the expected or mean value of an exponential distribution is a function of the parameter λ: [55]

$$\mathrm{E}[X] = \frac{1}{\lambda}. \tag{2.11}$$

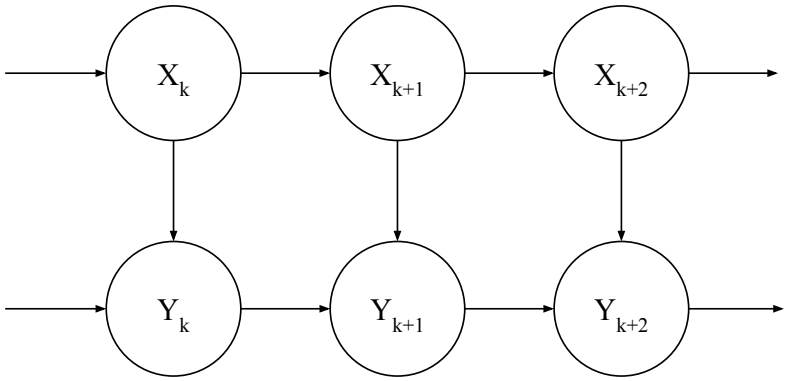

Figure 2.1: Behavior of a Hidden Markov Model. The model goes through a sequence of hidden states (X_k) and produces an observable output at each state (Y_k).

When there are multiple possible transitions from a state, each with their own expected transition time, the expected amount of time in the state is: [54]

$$\sum \lambda(x, y) = \sum \lambda p_{x,y} = \lambda(x) \tag{2.12}$$

where $\lambda(x, y)$ is the expected amount of time before state x transitions to state y. The expected time in a state ($\lambda(x)$) is related to the expected time for an individual transition ($\lambda(x, y)$) by a probability $p_{x,y}$. Each transition contributes to an expected amount of time in the state.

2.5.3. Hidden Markov Model. A Hidden Markov Model (HMM), pictured in Figure 2.1, is a Markov chain where the state is not directly visible to an observer. Instead, the HMM outputs observations related to the underlying, hidden chain. The hidden chain is assumed to meet the Markov property. A HMM can be described by the notation $\lambda = (\Pi, P, B)$, where Π is the initial state distribution vector, P is the matrix of state transition probabilities, and B is a matrix describing the probability of observing an output given the hidden state. Each row in B maps each state in X to a probability distribution for an observation from Y. Let y be some observation,

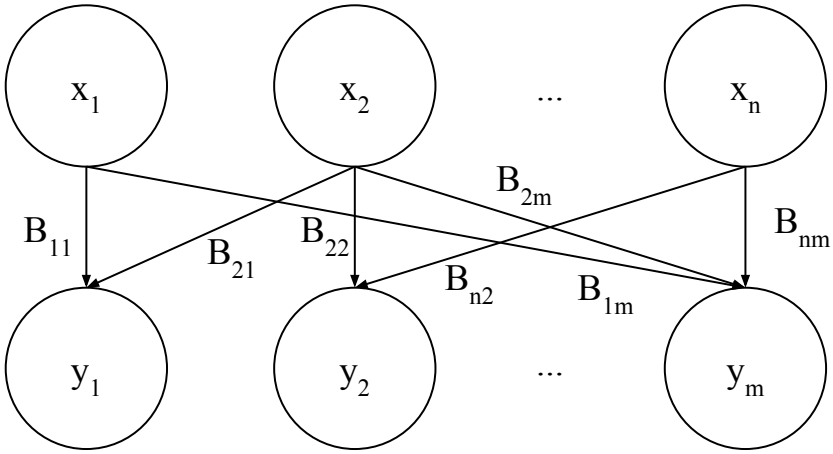

Figure 2.2: Observable output of a Hidden Markov Model. Each hidden state (x_i) has a corresponding probability distribution (B_i) for the observable outputs (y_j).

where $y \in Y$, a finite set of discrete, possible observations. B then is a $|X| \times |Y|$ matrix where $B_{ij} = \Pr(Y = y_j | X = x_i)$, the probability of observing y_j given the system is in some hidden state x_i. This relationship is pictured in Figure 2.2.

A model where the state is a combination of X and Y from a HMM is also a Markov model. The state space for the new model, Z, is a set of tuples $Z = \{(X_0, Y_0), (X_0, Y_1), ...(X_1, Y_0), ...(X_n, Y_m)\}$ where $\Pr(Z_k + 1 = (x_{k+1}, y_{k+1}) | Z_k = (x_k, y_k)) = \Pr(Y_{k+1} = y_{k+1} | X_{k+1} = x_{k+1}) \Pr(X_{k+1} = x_{k+1} | X_k = x_k)$. Stated plainly, the probability that the joint model is some state (x_{k+1}, y_{k+1}) is the probability that the hidden part of the model transitions to state x_{k+1} and the observation for x_{k+1} is y_{k+1}. The probability distribution for the transition to the next state does not depend on the y_k of the current state.

2.6. INFORMATION FLOW

2.6.1. Modal Logic. A Kripke frame is a pair $< W, R >$[27] such that W is a set of possible worlds, where each world corresponds to a unique global state of

the system. Each element of R describes a binary relationship for how the described system can move from world to world as events occur in the described system.

In the case of a distributed system, a world could be described as one of the possible combinations of values of all boolean state variables $S = \{s_0, s_1, ...s_n\}$ in the system. As execution occurs, messages, time, or events cause these variables to change. Each change in boolean variables corresponds to a relationship in R[49]. Therefore, a world w is one possible valuation of all the variables in S and a transition from w to another w' (with its own valuation) can be noted as wRw'. Without loss of generality, each relationship in R must result in the change of at least one variable in S. Additionally, the set of worlds is complete: every possible combination of state variable values is represented in the set of worlds. No relationship in R can lead to a world that does not exist.

Additionally, we can define a set of valuation functions, $\mathbb{V}$. Each function $V^i_{s_x}(w) \in \mathbb{V}$ describes the value observed by in a domain D_i of a boolean state variable s_x in some world w. If a valuation function for a particular state variable is not defined for an agent, the agent cannot determine the value of the state variable, and cannot determine the value of any logical statement based on the variable. In the case of a distributed system, the valuation function concept is analogous to the isolation of memory for each agent. For example, an agent i cannot simply determine the value of a variable for agent j.

The combination of a Kripke Frame $< W, R >$ and a set of valuation functions $\mathbb{V}$ is a Kripke model $K = \{W, R, \mathbb{V}\}$, frequently known as a modal model. The complete model describes all the possible worlds, the relation between those worlds and the information available in the domains of the system.

Let $\varphi \in \Phi_0$ be an atomic proposition in a set of countably many propositions. The set of Well-formed formulas (wffs), as defined by the formulation rules in Table 2.2, is the least set containing Φ_0. Additionally, we use the modal operator $\Box$ as an

Table 2.1: Logical Statement Formulation Rules

1. if φ is a wff, so are $\neg\varphi$, $\Box\varphi$, and $\Diamond\varphi$.
2. if φ is a wff, so are $B_i\varphi$ and $\neg B_i\varphi$
3. if φ is a wff, so are $T_{i,j}\varphi$ and $\neg T_{i,j}\varphi$
4. if φ is a wff, so are $I_{i,j}\varphi$ and $\neg I_{i,j}\varphi$
5. if φ and ψ are both wff, so are $\varphi \wedge \psi$
6. if φ and ψ are both wff, so are $\varphi \vee \psi$

abbreviation for $\neg\Diamond\neg\varphi$. The complete axiomatic system is outlined in Table 2.1. For the uninitiated, the modal box operator ($\Box$), "it is necessary that," states (in the case of $\Box\varphi$) "in every world w, φ is true." As its dual, the diamond operator ($\Diamond$) states "there is a world where φ is true."

2.6.2. Multiple Security Domain Model Non-Deducibility (MSDND). In the domain of security, there are a wide variety of aspects worth protecting in every system. These are grouped into the core security concepts of integrity, accessibility, and privacy. Many traditional security approaches rely heavily on cryptography to provide privacy. However, accidental information leakage can still occur, which compromises the privacy of the system. For CPS, the leakage is difficult to prevent. Unlike their purely cyber counterparts, the actions taken by the physical components cannot be easily hidden from an observer. For example, a plane changing altitude or a car turning or changing speed cannot be hidden from an observer. More complicated systems, like the power grid, have actions that are more difficult to observe. However, a well-motivated attacker can potentially collect critical information about the behavior of the cyber components with observations of the physical network[59].

Information flow security models are invaluable for assessing what information, if any, is leaked by either the cyber of physical components of the CPS. Many information flow security models, have been proposed, all based on similar concepts. Typically, the models partition the system into two domains: the high-security domain and the low-security domain. However, the MSDND security model allows the

Table 2.2: The Axiomatic System

Definition of logical and modal operators (abbreviations)

D1.	$\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$
D2.	$\varphi \oplus \psi \equiv (\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi)$ (exclusive or)
D3.	$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$
D4.	$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
D5.	$\Diamond\psi \equiv \exists w \in W : w \vdash \varphi$
D6.	$\Box\varphi \equiv \neg\Diamond\neg\varphi$
D7.	$B_i\varphi$ agent i believes the truth of φ
D8.	$I_{i,j}\varphi$ agent j informs i that $\varphi \equiv \top$
D9.	$T_{i,j}\varphi$ agent i trusts the report from j about φ

Axioms

P.	All the tautologies from the propositional calculus.
K.	$\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$
M.	$\Box\varphi \rightarrow \varphi$
A1.	$\neg\Box\varphi \rightarrow \Box\neg\Box\varphi$
A2.	$\Diamond(\varphi \vee \psi) \rightarrow \Diamond\varphi \vee \Diamond\psi$
A3.	$\Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$
B1.	$(B_i\varphi \wedge B_i(\varphi \rightarrow \psi)) \rightarrow B_i\psi$
B2.	$\neg B_i\bot$
B3.	$B_i\varphi \rightarrow B_iB_i\varphi$
B4.	$\neg B_i\varphi \rightarrow B_i\neg B_i\varphi$
I1.	$(I_{i,j}\varphi \wedge I_{i,j}(\varphi \rightarrow \psi)) \rightarrow I_{i,j}\psi$
I2.	$\neg I_{i,j}\bot$
C1.	$(B_iI_{i,j}\varphi \wedge T_{i,j}\varphi) \rightarrow B_i\varphi$
C2.	$T_{i,j}\varphi \equiv B_iT_{i,j}\varphi$

Rules of Inference

R1.	From $\vdash \varphi$ and $\vdash \varphi \rightarrow \psi$ infer ψ (Modus Ponens)
R2.	$\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$ (DeMorgan's)
R3.	From $\vdash \varphi$ infer $\vdash \Box\varphi$ (Generalization)
R4.	From $\vdash \varphi \equiv \psi$ infer $\vdash \Box\varphi \equiv \Box\psi$
R5.	From $\vdash \varphi \equiv \psi$ infer $\vdash T_{i,j}\varphi \equiv T_{i,j}\psi$

system to be partitioned into any number of domains. The MSDND model has been used to describe how the Stuxnet attack was able to hide its malicious behavior from the plant operators. The MSDND security model is expressed using modal logic to determine what information in a domain is deducible to an observer in another domain. The model exploits the possible worlds of modal logic to determine if there are worlds where the value of a logical atom is deducible by an agent outside the domain. As a consequence, it can be used to determine what an agent in a distributed system

can determine about another agent. The exact specification of timing the distributed system becomes unnecessary as the modal model can express any combination of logical atoms in one of its worlds.[36][25][37]

The MSDND security model can be expressed as follows[25]: Consider a pair of state variables s_x and s_y which may or may not be in the same security domain. The value of s_x and s_y have a logical xor relationship: if s_x is true, s_y must be false. Given an agent i that does not have a valuation function for either of those two variables, the system is MSDND secure for the agent and pair of variables. Written formally:

$$\begin{aligned} MSDND = \exists w \in W : w \vdash \Box[(s_x \vee s_y) \wedge \neg(s_x \wedge s_y)] \\ \wedge[w \vDash (\not\exists V_x^i(w) \wedge \not\exists V_y^i(w))] \end{aligned} \tag{2.13}$$

Of particular interest is the special case where s_x and s_y are relation on the same wff: ($s_x = \varphi$ and $s_y = \neg\varphi$):

$$\begin{aligned} MSDND = \exists w \in W : w \vdash \Box[\varphi \oplus \neg\varphi] \\ \wedge[w \vDash (\not\exists V_\varphi^i(w))] \end{aligned} \tag{2.14}$$

In a system where the above logical relationship holds, the agent i cannot determine the value of s_x or s_y. However, if the relationship does not hold, there is some world where the agent can determine the value of s_x and s_y.

2.6.3. BIT Logic. Belief, Information transfer, Trust (BIT) was developed to formalize logic about belief and information transfer. BIT logic has typically been applied to distributed systems but has also played roles in CPS security. The

operations of the BIT logic allow formal definition of how entities pass information, and how they will act on the information passed to them. BIT logic utilizes several modal operators:

- $I_{i,j}\varphi$ defines the transfer of information directly from agent j to an agent i.
- $T_{i,j}\varphi$ defines trust an agent i has in a report from j that φ is true.
- $B_i\varphi$ defines the belief that an agent i has about φ. The actual value of φ is irrelevant: the agent i believes it to be true.

These operators allow reasoning about information transfer between entities. In the context of a distributed system, these operators allow the division of the actual state held by some agent i to what some other agent j believes is agent i's state.

2.7. EXPLICIT CONGESTION NOTIFICATION

ECN is a technique for managing congestion in IP networks. When an ECN-capable network device detects congestion, it can drop the packets, or it can signal senders using flags in the packet headers that the network is congested. For a TCP application, the result of the dropped packets causes the slow-start congestion control strategy to reduce the rate packets are sent. A more advanced implementation, using ECN, sets specific bits in the TCP header to indicate congestion. By using ECN, TCP connections can reduce their transmission rate without re-transmitting packets.

UDP applications have not typically utilized ECN. Although the ECN standard has flags in the IPv4 header, access to the IPv4 header is not possible on most systems. Furthermore, there is not a "one size fits all" solution to congestion in UDP algorithms.

2.7.1. Random Early Detection (RED). The RED queuing algorithm is a popular queuing algorithm for switches and routers. It uses a probabilistic model

and an Exponentially Weighted Moving Average (EWMA) to determine if the average queue size exceeds predefined values. The values are used to identify potential congestion and manage it. Congestion identification is accomplished by determining the average size of the queue, and then probabilistically dropping packets to maintain the size of the queue. In RED, when the average queue size, avg, exceeds a minimum threshold (min_{th}), but is less than a maximum threshold (max_{th}), new packets arriving at the queue may be "marked". A packet is marked based on the following relation between p_b and p_a:

$$p_b = max_p(avg - min_{th})/(max_{th} - min_{th}) \tag{2.15}$$

$$p_a = p_b/(1 - count * p_b) \tag{2.16}$$

where p_a is the final probability a packet will be marked, max_p is the maximum probability a packet will be marked when the queue size is between min_{th}, and max_{th} and *count* is the number of packets since the last marked packet. With RED, the probability a packet is marked varies linearly with the average queue size and as a function of the time since the last packet was marked. If avg is greater than max_{th}, the probability of marking trends toward one as the average queue size approaches $2 * max_{th}$. In the event the queue fills completely, the RED queue operates as a drop-tail queue. In a simple implementation of the RED algorithm, marked packets are dropped.

2.8. DISTRIBUTED GRID INTELLIGENCE (DGI)

The DGI is a smart-grid operating system that organizes and coordinates power electronics. It also negotiates contracts to deliver power to devices and regions that cannot effectively facilitate their own needs. DGI leverages common distributed

algorithms to control the power grid, making it an attractive target for modeling a distributed system. Algorithms employed by the DGI and grouped into modules work together to migrate power from areas of excess supply to excess demand.

DGI utilizes several modules to manage a distributed smart-grid system. Group management, which is our main focus, implements a leader election algorithm to discover which processes are reachable within the cyber domain. Other modules provide additional functionality, such as collecting global snapshots, negotiating the migrations, and giving commands to physical components.

DGI is a real-time system; certain actions (and reactions) involving power system components need to be completed within a pre-specified time frame to keep the system stable. It uses a round robin schedule where each module is given a predetermined window of execution which it may use to perform its duties. When a module's round ends, the next module in the line is allowed to execute.

The DGI uses the leader election algorithm, "Invitation Election Algorithm," written by Garcia-Molina[29]. The algorithm provides a robust election procedure which allows for transient partitions. Transient partitions are formed when a faulty link inside a group of processes causes the group to divide temporarily. When the link becomes more reliable, the transient partitions merge.

2.8.1. Real Time. Real-time requirements were designed to enforce a strict upper bound on the amount of time used creating groups, discovering peers, collecting the global state, and performing migrations.

To enforce these bounds, the real-time DGI has distinct phases which modules were allowed to use for all processing. Each module was given a round with a specific amount of processor time allocated to the module. Modules used the allocated time to complete any tasks they had prepared. When the allotted time was up, the scheduler changed context to the next module. This interaction is illustrated in Figure 2.3.

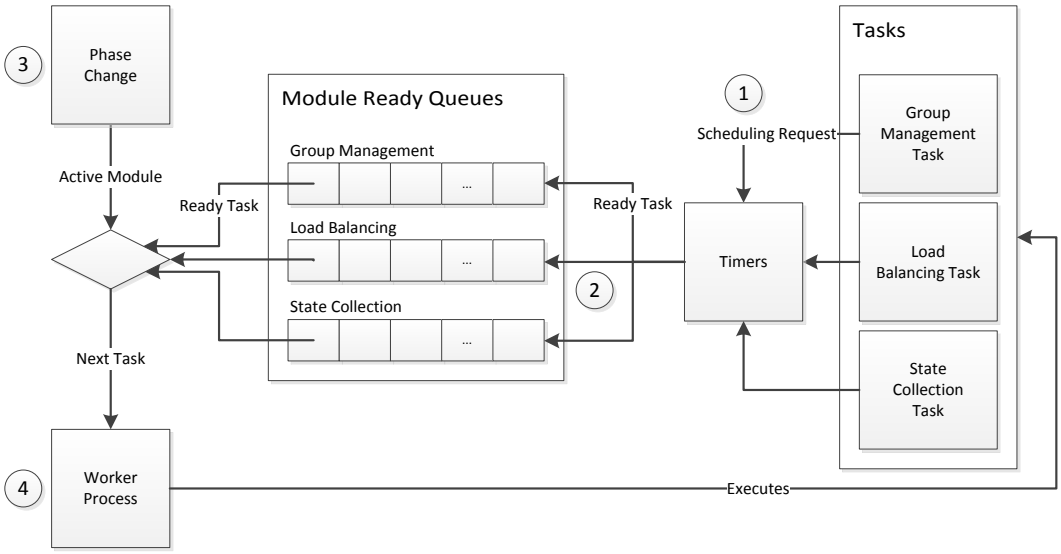

Figure 2.3: The real time scheduler used a round robin approach to allot execution time to modules. ① Modules requested a task be executed by specifying a time in the future to execute the task. ② A timer was set to count down to the specified moment. Modules could place tasks immediately into the ready queue if the task could be executed immediately. ③ When the timer expires, the task is placed into the ready queue. ④ Modules were assigned periods of execution (called phases) of a predetermined length. After the specified amount of time had passed, the module's phase ends and the next module in the schedule began to execute. ⑤ The worker selected the next ready task for the active module from the ready queue and executed it. These tasks could also schedule other tasks to be run in the future.

Modules informed the scheduler of tasks they wish to perform. The tasks could be scheduled for some point in the future, or scheduled to be executed immediately. When a task became ready, it was inserted into a ready queue for the module which scheduled the task.

When the module's phase was active, tasks were pulled from the ready queue and executed. When the phase was complete, the scheduler stopped pulling tasks from the previous module's queue and began pulling from the next module's queue.

Using a round robin scheduler allowed enforcement of an upper bound on message delay.

2.8.2. Group Management Algorithm. The DGI uses the leader election algorithm, "Invitation Election Algorithm," written by Garcia-Molina[29]. Originally published in 1982, the algorithm provides a robust election procedure that allows for transient partitions. Transient partitions are formed when a faulty link between two or more clusters of DGI causes the groups to divide temporarily. These transient partitions merge when the link becomes more reliable. The election algorithm allows for failures that disconnect two distinct sub-networks. These sub-networks are fully connected, but connectivity between the two sub-networks is limited by an unreliable link.

Since Garcia-Molina's original publication[29], a large number of election algorithms have been created. Each algorithm is designed to be well-suited the problem space where it is used. Specialized algorithms exist for wireless sensor networks[63][22], detecting failures in certain circumstances[66][51], and of course, transient partitions. Work on leader elections has been incorporated into a variety of distributed frameworks: Isis[11], Horus[58], Totem[52], Transis[5], and Spread[4] all have methods for creating groups. Despite the broad range of work, the fundamentals of leader election are consistent across all work. Processes arrive at a consensus of a single peer that coordinates the group. Processes that fail are detected and removed from the group.

The elected leader is responsible for making work assignments and identifying and merging with other coordinators when they are found, as well as maintaining an up-to-date list of peers for the members of its group. Group members monitor the group leader by periodically checking if the group leader is still alive by sending a message. If the leader fails to respond, the querying process will enter a recovery state and operate alone until they can identify another coordinator. Therefore, a leader

and each of the members maintains a set of processes which are currently reachable, a subset of all known processes in the system.

Leader election can also be classified as a failure detector[32]. Failure detectors are algorithms which detect the failure of processes within a system; they maintain a list of processes that they suspect have crashed. This informal description gives the failure detector strong ties to the leader election process. The group management module maintains a list of suspected processes which can be determined from the set of all processes and the current membership.

The leader and the members have separate roles to play in the failure detection process. Leaders use a periodic search to locate other leaders to merge groups. This query also serves to detect failures within the system. The member sends a query to its leader. The member will only suspect the leader and not the other processes in their group.

Using a leader election algorithm allows the FREEDM system to autonomously reconfigure in the event of a failure. Cyber components are tightly coupled with the physical components, and their reaction to faults is not limited to faults originating in the cyber domain. Processes automatically react to crash-stop failures, network issues, and power system faults. The automatic reconfiguration allows processes to react immediately to issues, faster than a human operator, without relying on a central configuration point. However, it is important that the configuration a leader election algorithm supplies is one where the system can do viable work without causing physical faults like voltage collapse or blackouts[16].

A state machine for the election portion of the election algorithm is shown in Figure 2.4. In the normal state, the election algorithm regularly searches for other coordinators. When another coordinator is identified, all other processes will yield to their future coordinator. The method of selecting which process becomes

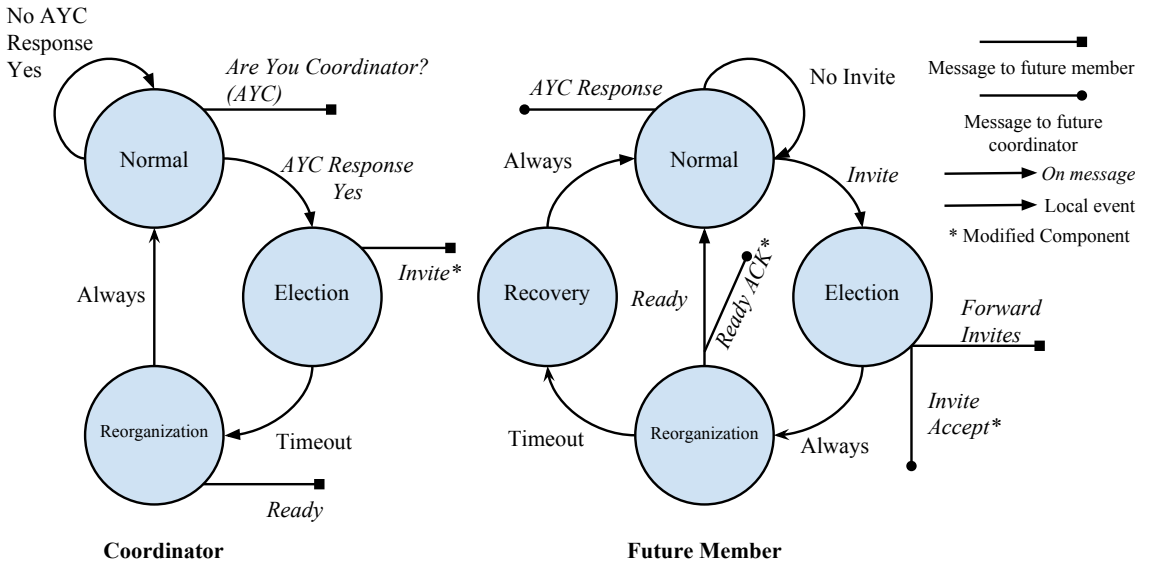

Figure 2.4: State machine for an election. Processes start as coordinators in the "Normal" state and search for other coordinators to join with. Processes immediately respond to AYC messages they receive. The algorithm was modified by adding a "Ready Acknowledgment" message as the final step of completing the election. Additionally, processes only accept invites if they have received an "AYC Response" message from the inviting process.

the coordinator of the new group differentiates the modified algorithm from other approaches.

2.8.3. Power Management. We utilized the load balancing algorithm from [3]. The load balancing algorithm performs work by managing power devices with a sequence of migrations[65]. In each migration, a sequence of message exchanges identify processes whose power devices are not sufficient to meet their local demand and other processes supply them with power by utilizing a shared bus. First, processes that cannot meet their demand announce their need to all other processes. Processes with devices that exceed their demand offer their power to processes that announced their need. These processes perform a three-way handshake. At the end of the handshake, the two processes have issued commands to their attached devices to supply power from the shared bus and to draw power from the shared bus. An

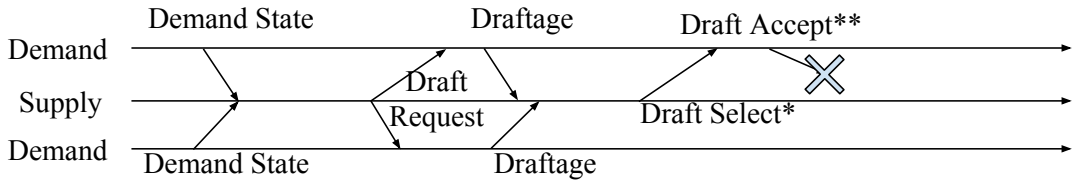

Figure 2.5: Example of a failed migration. (*) and (**) mark moments when power devices change state to complete the physical component of the migration. In this scenario, the message confirming the demand side made its corresponding physical change is lost, leaving the supply process uncertain.

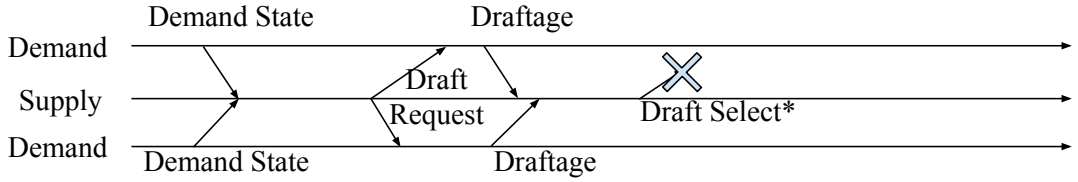

Figure 2.6: Example of a failed migration. (*) marks a moment when power devices change state to complete the physical component of the migration. In this scenario, the supply process changes its device state, but the demand process does not.

example of how the power system is affected by migrations is depicted in Figure 2.7. In the chart, processes with net power generation (generation > 0) share power with processes with excess loads. As processes with excess loads are satisfied, both supply and demand processes trend toward 0 net generation.

The DGI algorithms can tolerate packet loss and is implemented using UDP to pass messages between DGI processes. Effects of packet loss on the DGI's group management module have been explored in [38] and [39]. The load balancing algorithm can tolerate some message loss, but lost messages can cause migrations to only partially complete, which can cause instability in the physical network. A failed migration is diagrammed in Figures 2.5 and 2.6. With the power migration algorithm, uncompensated actions may occur in the power system. These actions can eventually lead to power instability through issues such as voltage collapse. Additionally, the supply process may not always be certain if the second half of the action was completed or not. If the "Draft Accept" message does not arrive from the demand

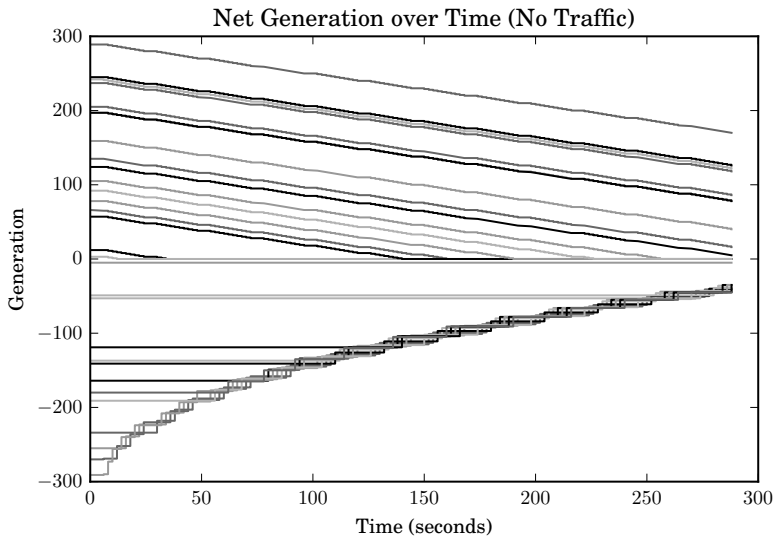

Figure 2.7: Effects of migrations on power. Each migration consumes excess generation capability and removes excess demand.

process, the supply process cannot be certain if its "Draft Select" message was received. If the supply process takes action to compensate by reversing the migration and the confirmation arrives later, the system will also be driven towards instability because another process completed an uncompensated action. Processes could confirm the number of failed migrations with a state collection technique. It is, therefore, desirable to manage the processes to minimize the number of failed migrations.

3. PROBLEM STATEMENT AND MOTIVATION

The entry point to research in this area was asking the question: "How does the software managing critical infrastructure, like FREEDM, behave when a critical component, the communication infrastructure, is not operating correctly?" Historically, leader elections have had limited applications in critical systems. However, in the smart-grid domain, there is a great opportunity to apply leader election algorithms in a directly beneficial way. [3] presents a simple scheme for performing power distribution and stabilization that relies on formed groups. Algorithms like Incremental Consensus Algorithm[70], begin with the assumption that there is a group of processes who coordinate to distribute power. In a system where 100% up time is not guaranteed, leader elections are a promising method of establishing these groups. A strong cyber-physical system should be able to survive and adapt to network outages in both the physical and cyber domains. When one of these outages occurs, the physical or cyber components must take corrective action to allow the rest of the network to continue operating normally.

This work observes the effects of message omission on the group management module of the Distributed Grid Intelligence (DGI) used by the FREEDM smart-grid project. This system uses a broker system architecture to coordinate several software modules that form a control system for a smart power grid. These modules include: group management, which handles coordinating processes via leader election; state collection, a module which captures a global system state; and load balancing, which uses the captured global state to bring the system to a stable state.

It is important for the designer of a cyber-physical system to consider what effects the cyber components will have on the overall system. Failures in the cyber domain can lead to critical instabilities which bring down the entire system if not

handled properly. In fact, there is a major shortage of work within the realm of the effects cyber outages have on CPS[64][68]. The analysis focuses on quantifiable changes in the amount of time a DGI could spend participating in energy management with other processes.

Using the DGI as a starting point, the analysis of the leader election algorithm in the DGI began with analysis of its behavior when messages were lost. To do this, the DGI software was subjected to omission faults while the state of the algorithm was captured over a period of examination. The goal of these experiments was to examine what behavior the DGI would exhibit during the fault conditions. Additionally, we hoped to determine the advantages and disadvantages of using a more complicated, reliable message retransmission protocol over a more simple one without retransmission.

3.1. INITIAL EXPERIMENTS

The initial experiments were collected from a non-real time version of the DGI code. Experiments measured the in group time (IGT), a measure of availability, for various sets of DGI running the leader election algorithm during omission fault conditions. Additionally, the experiments examined two communication modes, the Sequenced Reliable Connection, and the Sequenced Unreliable Connection. Both methods of communication were valid approaches for the message passing requirements of the DGI.

3.1.1. Sequenced Reliable Connection (SRC). The Sequenced Reliable Connection (SRC) is a modified send and wait protocol with the ability to stop resending messages and move on to the next one in the queue if the message delivery time is too long. When designing this scheme we wanted to achieve several criteria:

- Messages must be accepted in order - Some distributed algorithms rely on the assumption that the underlying message channel is FIFO.

- Messages can become irrelevant - Some messages may only have a short period in which they are worth sending. Outside of that time period, they should be considered inconsequential and should be skipped. To achieve this, we have added message expiration times. After a specified amount of time has passed, the sender will no longer attempt to write that message to the channel. Instead, it will proceed to the next unexpired message and attach a "kill" value to the message being sent, with the number of the last message the sender knows the receiver accepted.

- As much effort as possible should be applied to deliver a message while it is still relevant.

There was one adjustable parameter, the resend time, which controls how often the system would attempt to deliver a message it had not yet received an acknowledgment for.

3.1.2. Sequenced Unreliable Connection (SUC). The Sequenced Unreliable Connection (SUC) protocol is simply a best effort protocol: it employs a sliding window to try to deliver messages as quickly as possible. The receiver will look for increasing sequence numbers and disregard any message that is of a lower sequence number than is expected. The purpose of this protocol is to implement a bare minimum: messages are accepted in the order they are sent.

Like the SRC protocol, the SUC protocol's resend time can be adjusted. Additionally, the window size is configurable, but was left unchanged for the tests presented in this work.

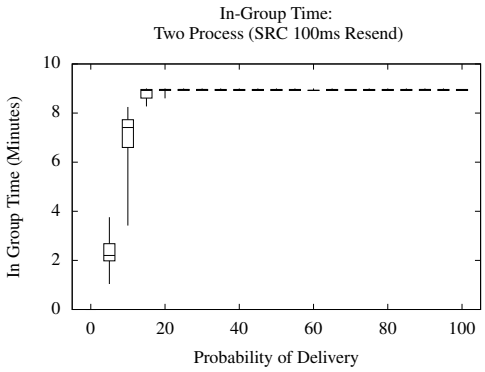

Figure 3.1: IGT over a 10 minute run for a two process system with a 100ms resend time.

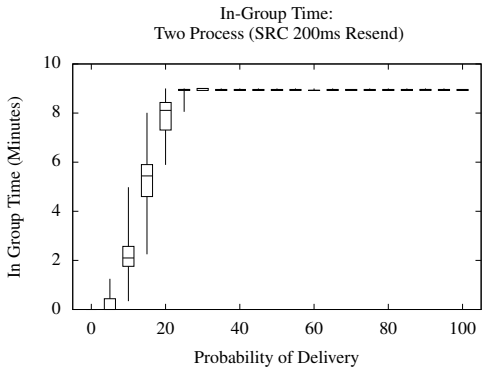

Figure 3.2: IGT over a 10 minute run for a two process system with a 200ms resend time.

3.2. INITIAL RESULTS

The collected results from the tests were divided into several target scenarios as well as the protocol used.

The first minute of each test in the experimental test was discarded so that any transients in the test could be removed. The tests were run for ten minutes, however the maximum result was 9 minutes of IGT. These graphs first appeared in [38].

3.2.1. Sequenced Reliable Connection.

Two Process Case. The 100ms resend SRC test with two processes was considered a type of control in this study. These tests, pictured in Figure 3.1, highlighted the availability of the DGI with the SRC protocol. The maximum IGT of 9 minutes was achieved with only 15% of datagrams arriving at the receiver.

Figure 3.2 demonstrates that as the rate at which lost datagrams were re-sent was decreased to 200ms, the in-group time decreased. This behavior was expected. Each exchange had a time limit for each message to arrive and the number of attempts was reduced by increasing the resend time.

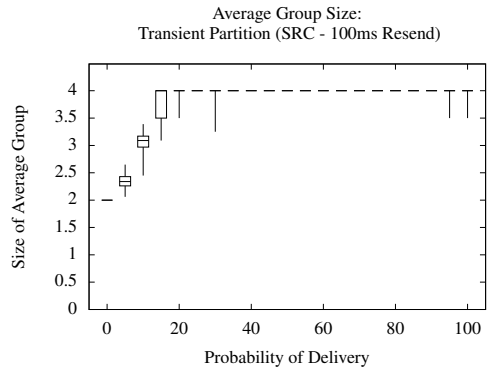

Figure 3.3: Average size of formed groups for the transient partition case with a 100ms resend time.

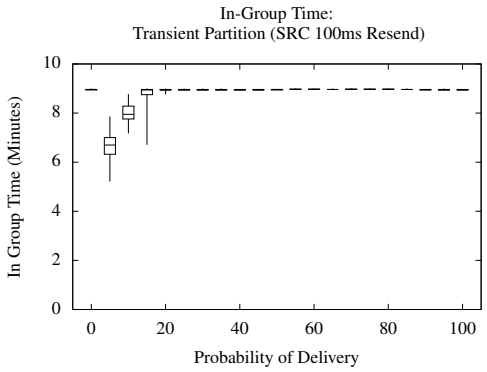

Figure 3.4: IGT over a 10 minute run for the transient partition case with a 100ms resend time.

Transient Partition Case. The transient partition case was a simple example in which a network partition separates two groups of DGI processes. In the simplest case, where the opposite side of the partition was unreachable, processes formed a group with the other processes on the same side of the partition. Two processes were present on each side of the partition. The 100ms case is shown in Figures 3.3 and 3.4.

While messages could not cross the partition, the DGIs stay in a group with the processes on the same side of the partition, leading to an in-group time of 9 minutes (the maximum value possible). As packets began to cross the partition (as the omission probability decreased), DGI instances on either side attempted to complete elections with the processes on the opposite partition and the in group time began to decrease. During this time, however, the mean group size continued to increase. Thus, while the elections were decreasing the amount of time that the module spent in a state where it can actively do work, it typically did not fall into a state where it was in a group by itself. As result, most of the lost IGT came from elections.

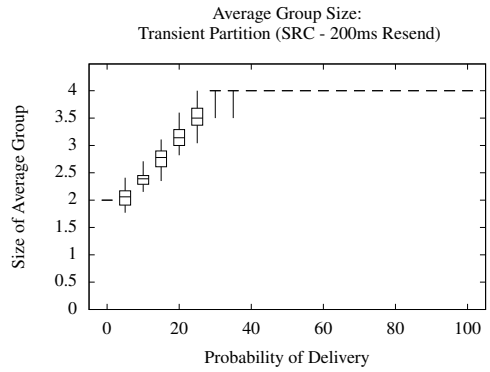

Figure 3.5: Average size of formed groups for the transient partition case with a 200ms resend time.

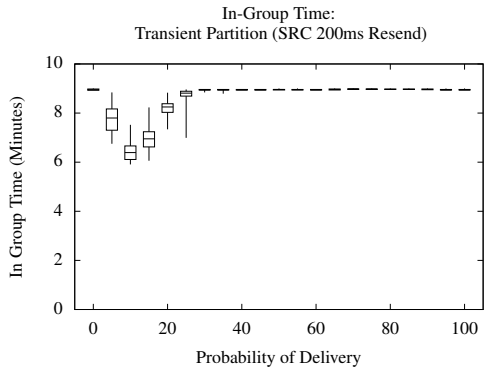

Figure 3.6: IGT over a 10 minute run for the transient partition case with a 200ms resend time.

The 200ms case (illustrated in Figures 3.5 and 3.6) suggests a similar behavior to Figures 3.3 and 3.4, with a wider valley produced by the reduced number of datagrams. The mean group size dipped below two in Figure 3.5, possibly because longer resend times allowed for a greater number race conditions between potential leaders. The race conditions are discussed during the SUC section since it was more prevalent in those experiments.

3.2.2. Sequenced Unreliable Connection.

Two Process Case. The SUC protocol's experimental tests revealed an immediate problem. There was a general increasing trend for the amount of IGT, shown in Figure 3.7. However, there was a high amount of variance for every trial.

In the 200ms resend case (illustrated in Figure 3.8), a greater growth rate occurred in the IGT as the omission probability decreased. When an average was taken across all of the collected data points from the experiment, the average IGT is higher for the 200ms case than it was for the 100ms case (6.86 minutes vs 6.09 minutes). There was a large amount of variance in the collected IGT, however. As a result, it is not possible to state with confidence that the there is a significant difference between the two cases.

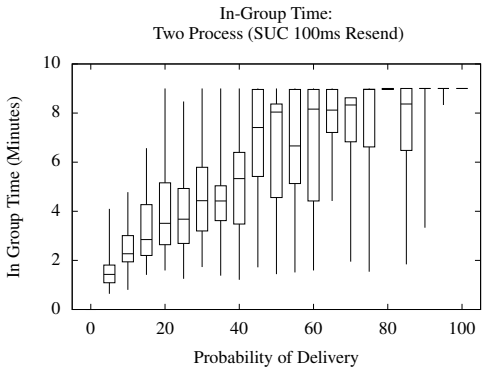

Figure 3.7: IGT over a 10 minute run for two process system with 100ms resend time.

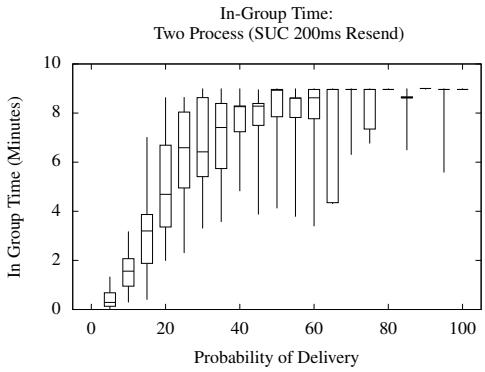

Figure 3.8: IGT over a 10 minute run for two process system with 200ms resend time.

3.3. MARKOV MODELS

After collecting the results from the initial experiments, we sought to describe the observed behavior through the use of continuous-time Markov chains. The behavior of the DGI transition between various states of grouping was calibrated with initial results and applied to other scenarios to validate the results. This approach had several shortcomings. First, for reasons we will demonstrate in subsequent chapters, the leader election algorithm modeled in these chains was not memoryless: the state used in the Markov chain was not sufficient to capture the interaction between processes that was occurring. Secondly, the continuous time model could not accurately capture the execution model of the DGI. In the DGI, processes synchronize with each other to execute in a partially-synchronous manner. As a result, the execution and transition between states is more accurately described with a discrete-time Markov chain.

3.3.1. Initial Model Calibration. The presented methodology of constructing the model was initially calibrated against the original two-process case. This calibration used a non-real-time version of the DGI code. The resulting Markov chain

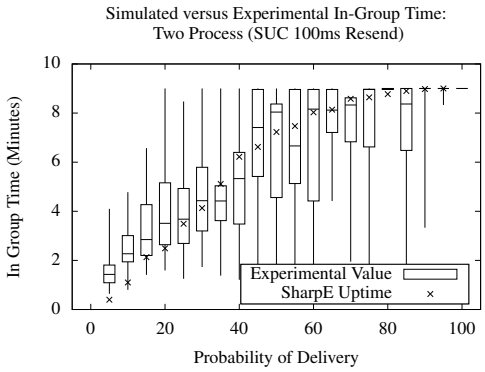

Figure 3.9: Comparison of in-group time as collected from the experimental platform and the simulator (1 tick offset between processes).

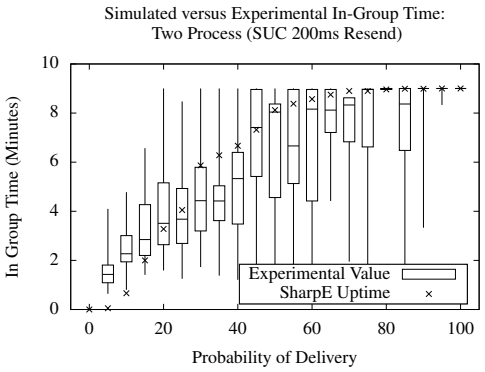

Figure 3.10: Comparison of in-group time as collected from the experimental platform and the simulator (2 tick offset between processes).

was processed using SharpE[44][60], a popular tool for reliability analysis. SharpE measured the reward collected in 600 seconds, minus the reward that was collected in the first 60 seconds. Discarding the reward from the first 60 seconds emulated the 60 seconds that were discarded in the experimental runs. The SharpE results are plotted along with the experimental results in Figures 3.9 and 3.10.

The race condition between processes during an election is a consideration in the original leader election algorithm, and is an additional factor here. The simulator provided a parameter to allow the operator to select how closely synchronized the peers were. The synchronization parameter was the time difference between when each process would search for leaders. The exchange of messages, particularly during an election, had a tendency to synchronize processes during elections. Processes could synchronize even if they did not initially begin in a synchronized state. The simulation results aligned best for the 100ms resend case with 1 tick (approximately 100ms difference in synchronization between processes) and 2 ticks (approximately 400ms) in the 200ms resend case.

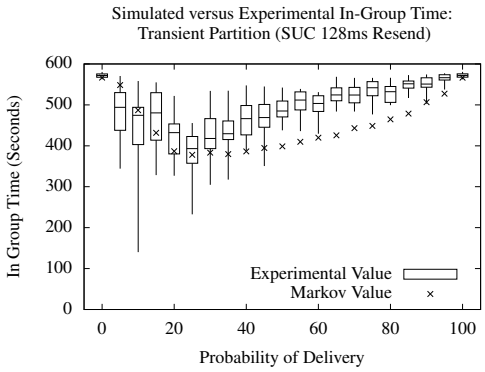

Figure 3.11: Comparison of in-group time as collected from the experimental platform and the in-group time from the equivalent Markov chain (128ms between resends).

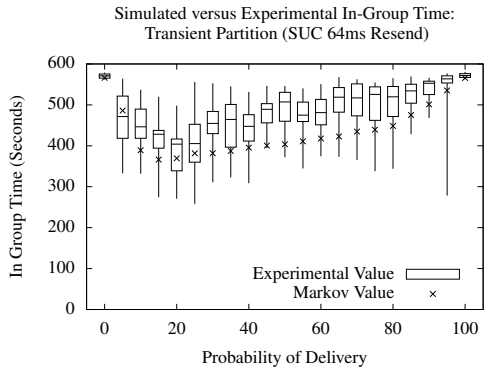

Figure 3.12: Comparison of in-group time as collected from the experimental platform and the in-group time from the equivalent Markov chain (64ms between resends).

Table 3.1: Error and correlation of experimental data and Markov chain predictions

Re-send	Correlation	Error
128	0.7656	11.61%
64	0.8604	11.70%

The structure of the Markov Chain assumed that processes enter the election state simultaneously. This was an appropriate assumption for the real-time system, since the round-robin scheduler synchronized when processes ran their group management modules. The simulator was set to assume that the synchronization between processes was very tight. New experimental data was collected for the 4 process, transient partition case. The collected data is overlaid with the results from the random walker in Figures 3.11 and 3.12.

As a measure of the strength of the model, the correlation between the predicted value was compared. The average error was also computed for each of the samples taken. This information is presented in Table 3.1. These results were not sufficient to accurately describe the behavior of the system during fault conditions.

As a result, we sought to refine the analysis of the model in order to get an accurate portrayal of the behavior of the system during faults.

3.4. REMARKS

To ensure the critical infrastructure can safely and reliably provides services to those needing the service, it is necessary to understand how the infrastructure behaves during faults. Conceptually, using unvetted critical infrastructure, especially when the infrastructure relies heavily on communication, is the same as stepping into an elevator without a safety brake. Strong analysis of a system's behavior during a failure scenario is paramount to ensuring the safety of those using the infrastructure.

We propose knowledge of the correctness of the operation may be more important than the efficiency of the operation of that critical infrastructure. Through this work we demonstrate how distributed algorithms can be improved by reasoning about them using information flow security models. Using these models, one can analyze where the certainty of the system is placed as well as ensuring that the algorithms can be modeled without complete and perfect information of the system.

The models created in this way allow a distributed critical infrastructure system to adjust its behavior, hardening itself against failures. With this hardening technique, the algorithms used by the system can prevent critical failures that could decrease quality of life for the people using that system. Subsequent chapters show how information flow methods can be used to determine the memorylessness of aspects of a distributed system, how those aspects can be used to construct a model, and applications of those models.

4. RELATED WORK

4.1. ANALYSIS OF DISTRIBUTED SYSTEMS

[20] focuses on examining the behavior of a collection of processes in a grid computing system processing a large dataset. In their work, the authors use a DTMC to model a single process completing a task. The chain describes how a process in the system goes through the steps of acquiring a task to work on, working on the task, and subsequently either completing or failing the task. The created models are "absorbing" chains, meaning it has one or more states the process cannot leave once it has arrived in those states. They consider the Markov chain as a max-flow min-cut problem using the "task complete" and "task failed" absorbing states as sinks. Their analysis uses minimal s-t cuts to determine critical paths for the ideal operation of the system. By identifying "critical transitions" in the graph, the edges that would most greatly affect the performance of the system can be identified.

[32] studies an Omega class failure detector using OmNet++[17], a network simulation software package. Instead of omission failures, however, it considers crash failures. Each configuration goes through a predefined sequence of crash failures, and OmNet++ is used to count the number of messages sent by each of three different leader election algorithms. Additionally, [32] only considers the system to be in a complete and active state when all participants have a consensus on a single leader.

[34] shows that in a distributed system, when there are omission faults, complete consensus cannot be achieved without access to "common knowledge." The authors use a logic system that isolates the atomic propositions (APs) a process can directly access to its memory space and a knowledge operator to access other APs when information is transferred between processes. It allows the authors to reason

about the necessary conditions to arrive at consensus when there are omission faults. They conclude consensus can only be reached when there is existing shared knowledge in a system that is both unaffected by the omission faults and is useful for arriving at a correct consensus. The authors establish, for the two armies problem, the generals would only reach consensus and attack successfully if it was common knowledge they would attack together. Additionally, they establish the common knowledge of the two generals attacking together could not be established in a system with omission faults.

4.2. PHYSICAL FAULTS CAUSED BY CYBER ENTITIES IN CPS

Faults in CPS can originate from many locations. First, and most obviously, the traditional physical system being augmented by the CPS is subject to its own failures, either from component failure or the actions of an attacker. Secondly, the CPS must employ sensors to detect the state of the physical components in the system. Like the physical components, these sensors are subject to component failure or the actions of an attacker. Lastly, if the CPS communicates between entities using a communication network, the network can be disrupted by any number of issues, including DoS or congestion caused by other users in a shared network.

Several works have shown[59][16][64] that for a computer-controlled smart-grid, failures originating at sensors or the communication network have the potential to cause the cyber entities controlling the physical networks to take incorrect actions. Approaches exist to identify faulty sensing components in a network, but the identification of bad sensing equipment may not always be possible. Additionally, it is not always possible to identify if the origin of an issue is a faulty sensor or an outside attacker.

If the cyber entity itself has been compromised, it could potentially exhibit Byzantine behavior, causing it to try and trick other components into bad actions[59], or it may try to disrupt the physical network directly. Work has been done to identify when an entity in a cyber network is actively working to compromise the physical network by using the underlying physical invariants. However, even if a cyber entity is trying to behave correctly, disruptions to the communication network or the sensors it uses can cause it to take actions similar to a process actively attempting to destabilize the system. Under the right circumstances, these actions can be identified by using the underlying invariants of the physical network. Ideally, however, the best goal is to avoid situations where a "good" entity is forced to act badly.

4.3. COMMUNICATION IN THE SMART-GRID

Communication in the smart-grid is still a rapidly developing area. In particular, because of the interest in the Internet of Things (IoT) and in Machine to Machine (M2M) communication, there are a wide range of communication protocols being developed for "smart" infrastructures like smart-cities, smart-factories, and the smart-grid. Historically, for the power grid, communication has been unidirectional. Because generation of power was centralized, communication requirements only necessitated communication paths between measurement points and control centers and from the control centers to individual substations[26]. The Supervisory Control And Data Acquisition (SCADA) systems used in the power grid were almost exclusively organized in a star topology where the individual devices had no means of communicating directly with each other.

In a smart-grid where power generation is widely distributed and citizens are encouraged to manage their energy usage for their own economic benefit as well as the planet's, the traditional SCADA design is not sufficient[26][33][43]. This version

of the smart-grid relies heavily on communication between devices, consumers, and the utility company to coordinate access to energy generated across a wide area. The internet has potential as the backbone for these communication requirements. Due to its prevalence, using existing internet infrastructure is an attractive, affordable option for realizing the requirements of a smart-grid[50][33]. Many companies in the power generation industry have expressed an interest in using the internet in the smart-grid[1].

Traditional SCADA systems for the power grid have used Distributed Network Protocol 3 (DNP3), a data-link layer protocol, to control substations and related devices[2]. For the smart-grid, IEC 61850 has emerged as a candidate for substation control[40][35][48]. IEC-61850 can be run over TCP/IP network or high speed switched LANs[48].

A number of potential attacks on SCADA-based control systems have been identified[67][7], including Denial of Service (DoS) attacks[47][41]. Regardless of the communication medium or the source of the issue, handling interruptions to the communication network in the smart-grid is an important concern. If a utility chooses to use a public network like the internet for any portion of its control network, that operator must consider the consequences of congestion on that network[24][28][69].

For congestion control in CPS, to our knowledge, only one other work has advocated the ECN approach: [15]. In [15], the authors propose varying the schedule and migration size of a load balancing-like algorithm in a network to maintain physical network stability during congestion. We adapted their measure "K", used to determine when the system would collapse due to incomplete migrations, to evaluate the success of our approach by counting the number of times migrations failed during the simulation.

5. INFORMATION FLOW ANALYSIS OF DISTRIBUTED COMPUTING

5.1. METHODS

A model created for a distributed system must have sufficient information to be accurate. The current state of a distributed system is difficult to obtain because memory spaces are typically isolated. Without exact synchronization, an accurate global snapshot of the system cannot be taken. Instead of attempting to capture exact global snapshots, our approach relies on allowing an agent to reason about the state of the other agents in the system. By doing so, an agent can construct a model. We propose the following for the execution environment of the distributed system:

- Each agent has some set of logical variables which it manipulates as its algorithms execute.
- Each agent belongs to a domain unique to the agent (agent i is the only member of the logical domain D_i).
- No agent can directly access a variable outside its domain.
- The authenticity of any information transfer (using modal operator $I_{i,j}$) is always trusted. However, the trust operator ($T_{i,j}$) is used to describe a message that is lost in transit: in all logical formulas presented, the trust operator describes the omission of a message.
- Agents do not exhibit Byzantine failure, nor do they crash; only messages may be omitted.

If no information is passed between agents, they are MSDND secure (ignoring any sort of leakage from interactions in the physical world). As information is passed,

aspects of the agent's state are leaked. However, depending on when messages are sent and which messages are lost, the agent can be left in doubt as to the state of the other agent. The two armies problem is a well-known thought experiment about the ability of multiple parties to reach consensus when there are omission faults.

5.2. TWO ARMIES PROBLEM

First, we will show that information flow analysis can be used to determine what state information is deducible to a particular agent in a system. We use the common two armies problem as a starting point. Using the MSDND security model, we will formally show which portions of the system state are leaked to an agent through message exchanges.

In the two armies problem, two agents, which are generals of their respective armies, must cooperate to attack an enemy city. However, the two armies are physically separated by the enemy city and must send messengers to coordinate their plan. Any messenger the generals send can be captured by the enemy city, preventing their message from being delivered. The attack will fail if the generals do not make an agreement on when to attack. The generals must come to an agreement when their channel for communication, a messenger, is unreliable.

After one message has been sent to one of the two generals, the state of the intended recipient is MSDND secure to the sender. Let φ_0 be an AP indicating "General A will attack at dawn."

Theorem 1. *If no messages are exchanged, the state of the two generals is mutually MSDND secure.*

Proof. If no messages are exchanged and no information is leaked from the physical world, the two generals have no way of determining the other's state. □

Theorem 2. *Once at least one messenger delivers a message to one of the generals, one of the generals is not MSDND secure.*

Proof. Let $\{\varphi_i : i \in 1, 2...n\}$ describe the state that a general has received φ_{i-1}.

Case 2.1. *One messenger is sent by General A and arrives at General B.*

If no confirmations are sent, then General A clearly cannot deduce if General B has received the message. To General A, General B is MSDND secure because General A has no way to valuate $B_B\varphi_0$. However, if B believes A's message, then A is not MSDND secure to B, because B believes that φ_0 is true.

1.	φ_0	General A decides to attack at dawn.
2.	$I_{B,A}\varphi_0$	General A sends a messenger to B informing them of their army's intent.
3.	$B_B I_{B,A}\varphi_0 \wedge T_{B,A}\varphi_0$	General B believes the message from general A.
4.	$B_B\varphi_0$	By C1.
5.	$B_B\varphi_0 \rightarrow \varphi_1$	General B knows the plan.
6.	$w \vDash V^B_{\varphi_0}(w)$	$V^B_{\varphi_0}(w)$ always returns true.

Therefore, A is not MSDND secure to B. However, $V^A_{\varphi_1}(w) \notin \mathbb{V}$, so B is secure to A.

Case 2.2. *Any number of messengers are sent and deliver their messages, alternating from General A or General B to the other general.*

As each messenger arrives, the receiving general will trust the integrity of the message and believe its contents, resulting in that general assigning value to φ_i.

1.	$B_B\varphi_0$	Continuing from Case 2.1.
2.	$B_B\varphi_0 \rightarrow \varphi_1$	General B decides to follow A's plan.
3.	$I_{A,B}\varphi_1$	General B sends a messenger to A informing them of their army's intent.
4.	$B_A I_{A,B}\varphi_1 \wedge T_{A,B}\varphi_1$	General A believes the message from general B.
5.	$B_A\varphi_1$	By C1.
6.	$B_A\varphi_1 \rightarrow \varphi_2$	General A agrees.
...		The same logical chain repeats.
7.	$w \vDash V^x_{\varphi_n}(w)$	$V^x_{\varphi_n}(w)$ is always true. x is A or B depending on the value of n.

Therefore, either A or B is not MSDND secure to the other for φ_n. In the case $n = 1$, B is now unsure that A has received φ_1 and cannot deduce if $B_A\varphi_1$. B is unsure of A's state and, as a consequence, A is MSDND secure to B. However, B is not MSDND secure to A because φ_1 is known to A. By extension, for $i = 2, 4...n$, B is secure to A, but not A to B. For $i = 3, 5...n$, A is secure to B, but not B to A. □

Theorem 3. *If a messenger is captured and the message is not resent, both agents will be secure on the last successfully delivered message φ_n or φ_0 if the first messenger is captured.*

Proof.

Case 3.1. *One messenger is sent and captured by the enemy.*

If the messenger does not arrive, it is equivalent to the messenger never being sent. (Theorem 1)

Case 3.2. *If $n-1$ messengers successfully deliver their message, but messenger n is captured, both are secure on φ_n.*

Suppose General A sends φ_{n-1} to B. On the delivery of the message φ_{n-1} to B, the value of φ_n is secure in B to A, as A has no way of knowing if φ_{n-1} was delivered, unless B sends φ_n with a messenger. When B does send φ_n, the messenger never arrives. As a consequence, General A has no way of assigning value to $B_A\varphi_n$ ($V^A_{\varphi_n} \notin \mathbb{V}$). However, as before, φ_{n-1} at A is not secure to B. □

5.3. BYZANTINE GENERALS

If General A is attempting to coordinate with multiple armies, the problem becomes more complex. If we extend the messenger analogy to cover faulty generals (ones sending incorrect information or omitting messengers), the generals can reach consensus if, for every faulty general, there are three generals that work correctly.[46]

Theorem 4. *In any message exchange that conforms to the constraints of the Byzantine Generals problem, all agents are MSDND insecure on the plan φ.*

Proof. Suppose agent i decides to use plan φ to attack and there is some set of Byzantine generals T and some set of loyal generals G ($i \in G$). If $|G| > 3|T|$, the algorithm executes successfully and $B_x\varphi : \forall x \in G$. Therefore, every general in G can valuate φ and the variable is insecure. □

However, in a general omission model, the level of loyalty from processes may be impractical to achieve. If the required number of "loyal" processes are not available, the system cannot achieve consensus.

5.4. ELECTION IN AN ANONYMOUS COMPLETE NETWORK

Consider a version of the "coin-flipping" leader election expressed in BIT logic. The algorithm is functionally identical, but contains additional guards in the conditionals as an expression of the agents' knowledge. Additionally, note the construct

of labeling each φ that an agent receives is simply assigning labels to make the algorithm easier to understand. The algorithm only considers the complete collection of φs for execution and not the source of any φ the algorithm uses.

Let ψ_i be the state where an agent i completes the algorithm as the leader, and γ_i as the state where an agent i has terminated the algorithm. The algorithm terminates when $\{\gamma_i = T : \forall i\}$ is satisfied and exactly one ψ_i is true ($\sum_i \psi_i = 1$).

Algorithm 1 Anonymous Coin Flipping Leader Election Expressed in BIT logic

1: $\varphi_i \leftarrow random(T, F)$
2: $\psi_i \leftarrow F$
3: Send φ_i to every active neighbor $(\forall j \neq i \ : I_j, i\varphi_i)$
4: Receive φ_j from every active neighbor
5: **if** $\varphi_i \wedge (\forall j \neq i, w \vDash V^i_{\varphi_j}(w) : \neg\varphi_j)$ **then**
6: $\quad \psi_i \leftarrow T$
7: $\quad \gamma_i \leftarrow T$
8: **else if** $(\varphi_i \wedge \exists j \neq i, w \vDash V^i_{\varphi_j}(\varphi_j)(w) : \varphi_j) \vee (\forall k \text{ s.t. } w \vDash V^i_{\varphi_k}(w) : \neg\varphi_k$ **then**
9: $\quad \varphi_i \leftarrow random(T, F)$
10: $\quad$ Go to next round
11: **else if** $\exists j \neq i, w \vDash V^i_{\varphi_j}(w) : \varphi_j$ **then**
12: $\quad \gamma_i \leftarrow T$
13: **end if**

Theorem 5. *Algorithm 1 may not terminate correctly unless there is detectable, perfect information transfer to all parties in the algorithm.*

Proof. Let i be the agent that would correctly terminate as the leader. Let j be a process that has selected $\varphi_j = F$.

1.	$\varphi_i = T, \varphi_j = F$	Initial conditions.
2a.	$I_{j,i}\varphi_i$	i sends φ_i to j.
2b.	$I_{i,j}\varphi_j$	j sends φ_j to i.
3a.	$\neg(B_j I_{j,i}\varphi_i \wedge T_{j,i}\varphi_j)$	j does not receive φ_i.
3b.	$B_i I_{i,j}\varphi_j \wedge T_{i,j}\varphi_i$	i receives φ_j.
4a.	$w \nVdash V_{\varphi_i} j(w)$	j cannot valuate φ_i.
4b.	$B_i\varphi_j$	By C1.
5a.	$\nexists V^j_{\varphi_i}(w) \wedge \neg\varphi_j \rightarrow (\varphi \leftarrow T)$	j Cannot determine that i can terminate and incorrectly changes φ.
5b.	$\varphi_i \wedge w \vDash V^i_{\varphi_j}(w) \wedge \neg\varphi_j \rightarrow \psi_i$	i incorrectly terminates as the leader.

□

Therefore, i will terminate before j decides to go passive. This result agrees with results from similar analysis[21].

5.5. MODEL CONSTRUCTION FOR THE TWO ARMIES PROBLEM

Suppose for the two armies problem, A and B wish determine their chance of success. If neither A nor B have any idea of the probability their messenger is captured, they have no way of assessing their chances of a successful attack. However, what can be determined if the two generals share some knowledge about the world?

Suppose that both generals know their messenger has the probability $q = (1 - p)$ of being captured in transit. When A sends a messenger to B, A will know there is a probability p that the messenger arrives at B. If the messenger does indeed arrive at B, B knows that A knows there is a p chance their attack will succeed. In this situation, both generals can construct identical models of the success the attack, given the initial message arrives. It is also worth noting in this scenario, no additional

messages can improve the outlook of the two generals. Each additional message only confirms there is a probability p the other general will also attack.

5.5.1. Generalization. The idea of converting a Kripke model to a Markov model is mentioned in [42] and [62]. However, we could not locate a formal description of the method, so we present one here: given a Kripke Model $K =< W, R, \mathbb{V} >$, where we obligate K to contain sufficient state information to ensure the probability mapping is memoryless. Let $f_W : W \to X$ be a bijective mapping from W to X, where X is the set of states for a Markov model. Let f_R be a mapping for a relationship $wRw' \in R$ to $\Pr(X_{i+1} = f_W(w')|X_i = f_W(w))$. Therefore, for a Kripke-Markov model $P_{ij} = f_R(w, w')$ where $i = f_W(w)$ and $j = f_W(w')$.

For a process i, information about the system state is restricted to those APs in its domain, D_i. Let $O : D \times X \to Y$ be a surjective mapping from the complete, hidden system state to an observable state for a domain. With O, one can construct B, the observation probability distribution matrix for a HMM:

$$B_{ij} = \begin{cases} 1, & \text{if } O(D_i, x_i) = y_j \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Intuitively, one or more worlds will look identical to a process i, since the information necessary to differentiate them is not in D_i. As a side effect of O being a deterministic, surjective mapping, a special case exists where the observed state has the Markov property:

$$\Pr(Y_{k+1}|Y_k) = \sum_{\{X_{k+1}|O(X_{k+1})=Y_{k+1}\}} \Pr(X_{k+1}|X_k), \forall X_k \in \{X_k|O(X_k) = Y_k\}. \tag{5.2}$$

The observed probability of going from Y_k to Y_{k+1} is the probability that the hidden state X_k transitions to a state X_{k+1} such that the observation for X_{k+1} is

Y_{k+1}. It must also hold that the observed probability does not depend on the hidden state: any X_k that results in an observation Y_k must have the same probability of transitioning to Y_{k+1} as the other X_ks that yield the same observation.

5.5.2. State Determination. When an agent uses the information transfer operator ($I_{i,j}$) to pass information to another agent in the system, it intends for that agent to believe the passed wff. When an agent distributes a wff to many agents with the information transfer operator, it leads to a set of beliefs about the beliefs of the receiving agents. The set, N_i, is the set of beliefs agent i can have if all the wff it passed to the other agents are believed. For example, if an agent distributes a wff φ to a set of agents Ag ($i \notin Ag$), then $N_i = \{B_i B_j \varphi : \forall j \in Ag\}$. Since the belief of each $B_j\varphi$ is outside of the domain D_i, the agent i can only valuate a wff in N_i which has been leaked to i.

Let the set L_i be the subset of N_i for which a valuation function exists in a domain i. L_i can be populated either by direct information transfer or information leakage from interactions with agents. We can similarly define a set M_i which is the subset of N_i and superset of L_i. The values in M_i correspond to the beliefs of agent i if i had perfect knowledge of the beliefs other agents ($L_i \subseteq M_i \subseteq N_i$).

Theorem 6. *Each member of L_i is MSDND insecure to i.*

Proof. Each wff in L_i has a valuation function in the domain i.

1.	$I_{j,i}\varphi$	i informs some agent j of φ.
2.	$B_j I_{j,i}\varphi \wedge T_{j,i}\varphi$	j receives φ and believes its authenticity.
3	$B_j\varphi$	By C1.
4.	$I_{i,j}\varphi_{ack}$	j acknowledges $B_j\varphi$.
5.	$B_i I_{j,i}\varphi_{ack} \wedge T_{i,j}\varphi_{ack}$	i receives φ_{ack}.
6.	$B_i\varphi_{ack}$	By C1.
7.	$B_i\varphi_{ack} \rightarrow B_i B_j\varphi$	Because j acknowledged φ, i believes j believes φ.
8.	$w \vDash V^i_{B_i B_j\varphi}(w)$	i does believe φ.
9.	$w \vDash V^j_{B_j\varphi}(w)$	Is always true.

Therefore, $j \in L_i$, and $B_j\varphi$ is MSDND insecure to i.

□

Theorem 7. *Each wff in M_i and N_i but not L_i is MSDND secure to i.*

Proof. Each wff in M_i and N_i have no valuation in the domain i.

Case 7.1. *The case where j receives some wff φ and is in M_i but not L_i.*

1.	$I_{j,i}\varphi$	i informs some agent j of φ.
2.	$B_j I_{j,i}\varphi \wedge T_{j,i}\varphi$	j receives φ and believes its authenticity.
3	$B_j\varphi$	By C1.
4.	$I_{i,j}\varphi_{ack}$	j acknowledges $B_j\varphi$.
5.	$\neg(B_i I_{j,i}\varphi_{ack} \wedge T_{i,j}\varphi_{ack})$	i does not receive φ_{ack}.
6.	$w\neg \vDash V^i_{\varphi_a ck}(w)$	i is uncertain if $B_j\varphi$.
7.	$w \vDash V^j_{\varphi}(w)$	Is always true.

Therefore, $j \in M_i$, and $B_j\varphi$ is MSDND secure to i.

Case 7.2. *The case where j does not receive some wff φ and is in N_i but not M_i.*

1.	$I_{j,i}\varphi$	i informs some agent j of φ.
2.	$\neg(B_j I_{j,i}\varphi \wedge T_{j,i}\varphi)$	j does not receive φ.
3.	$w \nVdash V^i_{B_j\varphi}(w)$	i is uncertain if $B_j\varphi$.
4.	$w \nVdash V^j_{\varphi}(w)$	j is uncertain of φ.

Therefore, $j \in N_i$, and $B_j\varphi$ is MSDND secure to i.

□

We assumed any beliefs an agent held stem from information transfer from another agent. Therefore, we can assert the beliefs in L_i for any process i must have a traceable history derived from a process having a valuation for the referenced wff that aligns with the belief process i holds about the wff. Furthermore, wff in N_i but not L_i do not have valuation in domain D_i and cannot be used in an observation.

6. ALGORITHM AND MODEL CREATION

6.1. ORIGINAL INVITATION ELECTION ALGORITHM OVERVIEW

A state machine for the election portion of the invitation-election algorithm is shown in Figure 2.4. In the normal state, the election algorithm regularly searches for other coordinators to join. When another coordinator is identified, the identifying coordinator will attempt to invite the other coordinator to their group. In the invitation election algorithm, processes are assigned a priority based on their process ID. The coordinator with the highest priority is the first to send invites. After a brief delay, if it appears that coordinator did not send their invites, the next highest process will send their invites. If a coordinator accepts an invitation, it will forward the invite to its group members. Processes that receive an invite that are not already in an election will accept the invite. Once a timeout expires, the coordinator will send a "ready" message with a list of peers to all processes that accepted the invite. The invited processes have timeouts for when they expect the ready message to arrive. If the message does not arrive in time, the process will enter the recovery state where it resets to a group with itself as the only member.

Once a group is formed it must be maintained. To do this, processes occasionally exchange messages to verify the other is still reachable. The interaction is shown in Figure 6.1. Coordinators send "Are You Coordinator" messages to members of its group to check if the process has left the group. Group members send "Are You There" messages to the coordinator to verify they haven't been removed from the group and to ensure the coordinator is still alive. If processes fail to reply to a received message before a timeout, they will leave the group. Leaving the group can

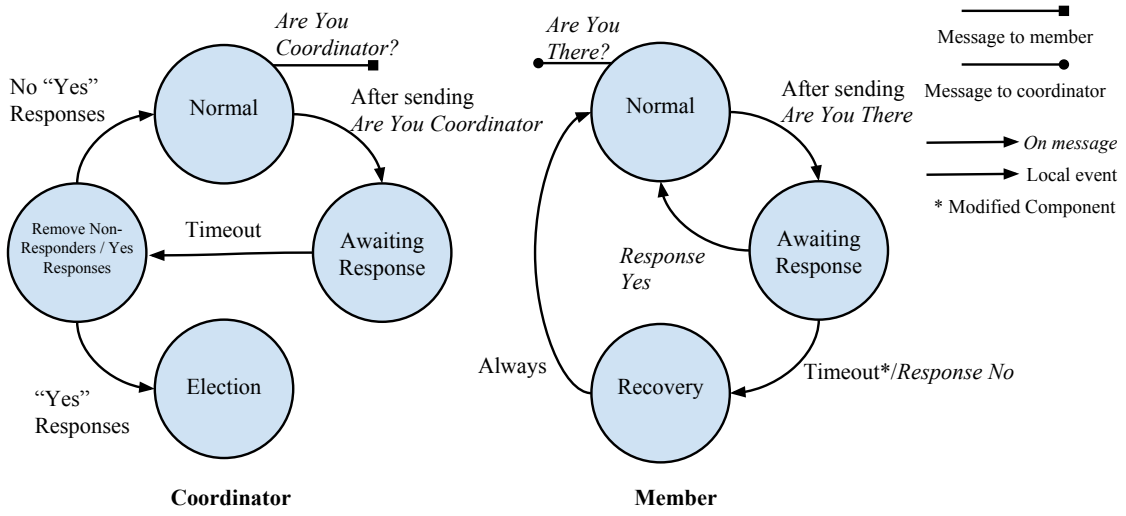

Figure 6.1: State machine for maintaining a group. The AYC messages are the same as those in Figure 2.4. AYC and AYT are periodically sent by processes, and responses to those messages are immediately sent by the receiving process. In the modified algorithm, the member does not enter the recovery state if they do not receive an AYT response before the timeout expires.

initiated by the coordinator removing the process, or the process can enter a recovery state and leave the group, forming a new group with itself as the only member.

6.2. EXECUTION ENVIRONMENT

Execution occurred in a real-time, partially synchronous environment. The execution environment was subject to omission failures. Processes synchronized their clocks and executed steps of the election algorithm at predefined intervals. Processes with clocks that were not sufficiently synchronized could not form groups. For this work, process execution occurred in an environment where the clocks were sufficiently synchronized to consistently form groups. In the real-time environment, messages that were delayed and missed their real-time deadlines had the same appearance as an omitted message. The execution environment for the election algorithm had an

omission fault occurrence modeled as a Bernoulli trial. In this model, each message had some probability p of being delivered within the timing constraints imposed by the real-time schedule. For the purpose of analyzing the effects of omission failures, processes were not subject to other faults.

6.3. MODEL CONSTRUCTION

A process can construct a Markov chain if its observations are consistent with the transition probabilities the process can determine with its observable state. In each election, a process has some probability of joining a given process' group. Let $\Pr(i \to j)$, be the probability the outcome of an election is i being a member of j's group. Since every process must be a member of some group:

$$\sum^{j} \Pr(i \to j) = 1. \tag{6.1}$$

Let $\Pr(i \to j | i \in k)$ be the probability i joins j at the end of an election given i starts in k's group at the start of the election. A similar relationship to Equation 6.1 must exist for this circumstance:

$$\sum^{j} \Pr(i \to j | i \in k) = 1. \tag{6.2}$$

A leader process j can construct a Markov chain iff, for every i that can join j's group:

$$\Pr(i \to j | i \in k) = \Pr(i \to j), i \neq j \neq k \tag{6.3}$$

where $i \in k$ is a state where i is in k's group. Therefore, the probability of i joining j's group must be independent of i's membership at the start of the election. Equation 6.3 enforces the probability j observes some process i joining its group is independent

of i's previous membership, which is hidden to j unless i was in j's group. From Equations 6.1 and 6.2:

$$\sum_{n \neq j} \Pr(i \to n | i \in k) = \sum_{n \neq j} \Pr(i \to n). \tag{6.4}$$

If more than one process can create a Markov chain, an algorithm which allows for that circumstance will have undesirable behavior. Suppose there are three processes A, B, and C. Without loss of generality, assume C is in A's group. For B to be able to construct a Markov chain, it must be equally likely that C leaves A's group to join B's as it is likely C would join B given it was not in A's group. In a system with no omission faults, ideally, a process will join a group and never leave it, especially if the election algorithm is being used to find a consensus value. Therefore, we restrict model construction to the highest priority process.

6.4. MODIFIED ELECTION ALGORITHM

The complete, modified election algorithm is listed in the appendix.

6.4.1. State Determination. In the Garcia-Molina version of the algorithm, processes distribute a list of processes that have accepted their invite. Let i be the coordinator of a group distributing ready messages. The process i has a list of processes who have accepted its invite. Let φ_x be a wff indicating some process x is part of i's group. Let $G = \{\varphi_x | x \in AcceptedInvite\} \cup \{\varphi_i\}$ be the set of processes that have accepted i's invite. To finalize the group, i will distribute G to each process described in G. If a processes does not receive G it will not be in the group.

Theorem 8. *In the original algorithm, each process is MSDND secure on φ_x for each x that is not i or itself (j).*

Proof.

Case 8.1. *The case where $x = i$ or $x = j$.*

If we assume the authenticity of messages is not questioned in this environment, a ready message from i must mean i is a part of the group. Therefore, i's inclusion in the group is MSDND insecure to j.

If j accepts the ready message, that process will be a part of the group in G and since j knows its own state, j knows it is a part of G.

Case 8.2. *The case where $x \neq i$ and $x \neq j$.*

For process i, i distributes G to each other process in G. Let Q be some set of processes that do not receive the G set.

1.	$I_{j,i}G\forall j \in G$	i distributes the list of processes that have accepted the invite.
2.	$\neg(B_x I_{x,i}G \wedge T_{x,i}G)\forall x \in Q$	Processes in Q do not receive G.
3.	$B_x I_{x,i}G \wedge T_{x,i}G\forall x \notin Q$	Processes not in Q receive G.
4.	$w \nvdash V^j_{B_x G}(w)\forall\{x \mid x \notin \{i,j\}\}$	j does not know which processes received G

Therefore, the receipt of G by other members of the group is MSDND secure to j.

□

Corollary 9. *The set of processes in the ready message is an N_i set, where i the group's coordinator.*

Because every wff in G is MSDND secure, G must be an N_i set, and $L_i = \emptyset$. As a consequence, the group leader only has an estimate of the system, which may be an overestimate ($M_i \subseteq N_i$).

With a ready acknowledgment message from members of the group, the leader can construct an L_i set, which underestimates the state of the system. From the previous chapter, we have shown that with message passing and omission, at least one process is left insecure to the other.

Theorem 10. *If a process j sends a ready acknowledgment message to i, j's inclusion in the i's view of the system state is MSDND secure to j.*

Proof. When j sends the ready acknowledgment message to i, it is uncertain if the acknowledgment message is actually delivered. From Theorem 2.1, we know that j is now MSDND insecure to i, but i is secure to j. Using the ready acknowledgment messages, i can then valuate $B_i B_j \varphi_j$ for every j sending an acknowledgment message. □

Once the ready acknowledgment messages have been sent to the leader, the group can begin to interact. As it is impossible for the members of the group to be distributed the L_i set, they must operate using the N_i set they received from the coordinator. Messages from other processes which could only have been sent if they are a part of the M_i set can leak to the receiving processes their receipt of the G set from i. For the purpose of model construction, the determination of members of M_i that are not in L_i may be undesirable for a DTMC since it may involve changing the state captured for the chain at the incorrect moment. Instead, for simplicity, while the leader can update their L_i set with leaked information, we avoid doing so to preserve the memorylessness property in the following sections.

6.4.2. Memorylessness. A process can update L_i with leaked information. To ensure the memorylessness property, we are particularly interested in the situation where a process receives the ready message, but the acknowledgment is lost, meaning the process is not included in the L_i set. We wish to ensure that the process can determine its own exclusion in L_i so it can behave as though it is not in M_i. To do this, we attach an additional field to the AYC message to indicate if the coordinator i considers the receiving process j a part of the L_i set. The receiving process can infer it is not part of L_i and it should behave accordingly.

Likewise, for the coordinator, an AYT message leaks that the sending process is a part of the M_i set if it is not in L_i. Since the behavior of the process in M_i is defined to behave as though it was not in L_i, the coordinator should do the same. This should be the case even if the process sending AYT has already responded negatively

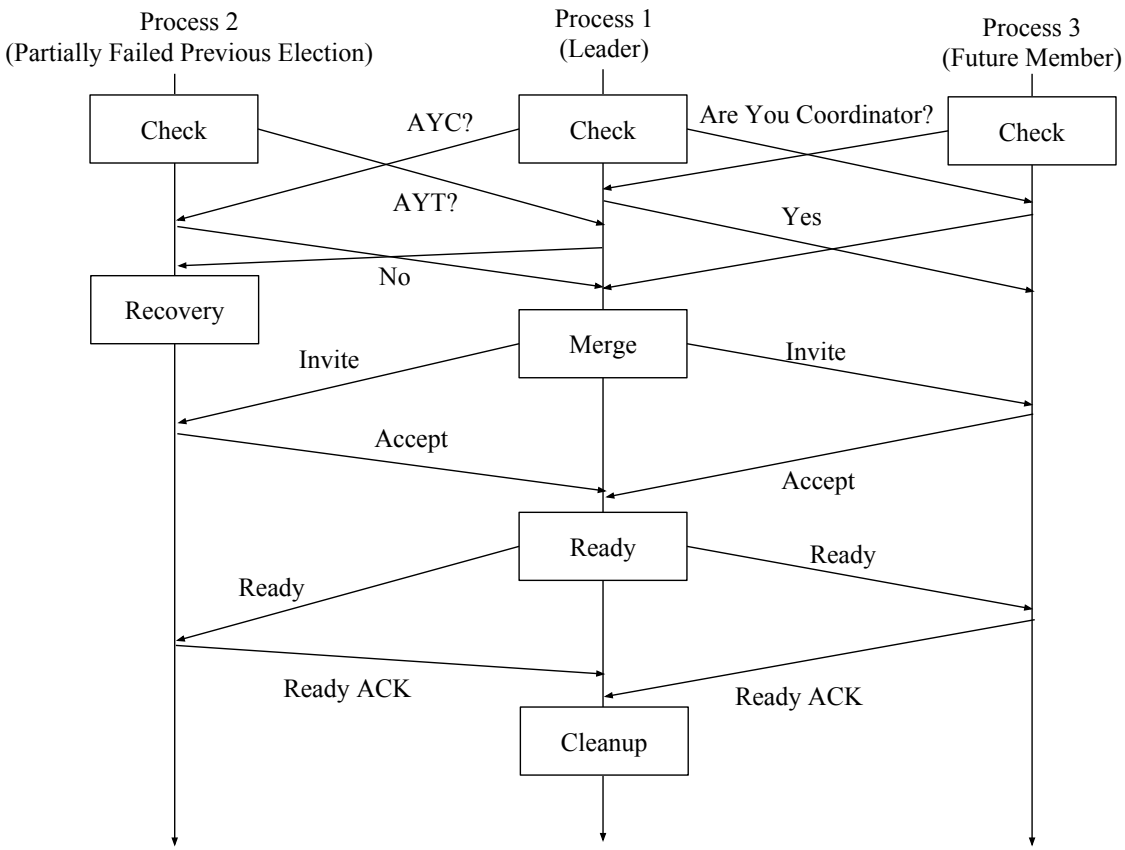

Figure 6.2: Diagram of message exchanges for an election. Process 2 almost completed the previous election and considers itself a part of process 1's group, but process 1 does not. Process 3 does not consider itself a part of process 1's group before the election begins. Both process 2 and process 3 need to successfully exchange the same number of messages to successfully complete the election.

to an AYC from i. For the purpose of memorylessness, i should consider j to have responded in the affirmative.

Processes determine which invitation to accept based on the exchange of the AYC messages. Processes always seek to reach the "lowest energy state": they wish to be in a group led by the coordinator with the highest priority. Processes will submit AYC messages to higher priority processes that are not its coordinator, even if they are in a group. As a result, the processes will always seek highest priority, regardless of their current group. More importantly, this releases the process from obligations on its next state from the state of other processes in the system. A process will only accept a received invitation if it has determined the sending process is the highest priority process.

In Figure 6.2, we diagram the sequence of message exchanges and function calls for two possible states for a process not in the leader's group. In the first state, the process nearly completed an election the previous round, but the ready acknowledgement message was not delivered in time, resulting in the process being in the M_i set. In the second state, the process is not in the leader's group. Both cases require the same number of messages to be delivered for the process to be considered a part of the group.

Therefore, regardless of the hidden state of the system, the highest priority process' observations will produce a Markov chain. The probability of transitioning to the next state is not influenced by the interactions of the other processes: it only depends on the combinations of outcomes based on the current observable state.

Theorem 11. *The observations of the highest priority process are sufficient for producing a Markov chain from the underlying HMM.*

Proof.

Case 11.1. *The system has one other process ($n = 1$).*

If there is only one other process, the probability of observing a group of two processes given the current state is a direct observation.

Case 11.2. *The system has two other processes ($n = 2$).*

In this case, there are 3 possible states for each of the other processes to be in: grouped with the highest priority process, ungrouped, and grouped without the highest priority process. When the other processes are grouped with the highest priority process, the probability of the other processes remaining in the group depends on the probability the current grouping is maintained. In the second scenario, where processes are ungrouped, the probability only depends on the probability they successfully contact the highest priority process. The last scenario is identical to the second, as the processes will behave as though they are ungrouped when contacted by the highest priority process.

The only observation that can be made by the highest priority process is the inclusion or exclusion of the other processes in their group. The probability of the processes excluded in the group joining the group is independent of their current state. Let X_G be the set of states for the HMM where the other processes are excluded from P_0's group, and in their own group led by P_1. X_U is the set of states where the processes are excluded from P_0's group and are themselves ungrouped. By the nature of the algorithm:

$$\Pr(O(D_0, X_{k+1})|X_k \in X_G) = \Pr(O(D_0, X_{k+1})|X_k \in X_U). \tag{6.5}$$

As a result, the ungrouped and grouped sub-cases have the same probability of resulting in a particular Y_{k+1}. Furthermore, based on the observations of the ungrouped process:

$$O(D_0, x_i) = O(D_0, x_j), \forall x_i \in X_G, \forall x_j \in X_U. \tag{6.6}$$

The equation indicates every observation of a state in X_G and X_U looks identical to P_0. With this restriction:

$$\Pr(Y_{k+1}|Y_k) = \sum_{\{X_{k+1}|O(X_{k+1})=Y_{k+1}\}} \Pr(X_{k+1}|X_k), \forall X_k \in \{X_k|X_k \in X_U \cup X_G\}. \quad (6.7)$$

Where $X_U \cup X_G$ is the complete set of states for processes not in P_0's group.

Case 11.3. *The system has a number of other processes ($n > 2$).*

Since the interactions of the other processes do not affect the probability of joining P_0's group, the previous case directly extends to this case.

□

6.4.3. Model Construction. Based on the state determination and memorylessness arguments presented, the highest priority process operates independent of the state of the other processes in the system. The structure of the algorithm prevents any process from being locked out of participating in a round of execution based on the outcome of the previous round. Therefore, for every observation of the system state the highest priority process makes, the next round of execution for all the other processes favors the high priority process, ensuring that its observation corresponds directly with the system state.

The states of the Markov chain are the cardinality of the highest priority process' group. In each round, the behavior is described by two components: maintaining a group and inviting other processes into the group. The coordinator will exchange an "Are You Coordinator" message and the peer will respond to verify it is still available. To maintain a group of m other processes, the probability is defined as a random variable with the following pdf:

$$\Pr_M(Y = k; m) = \begin{cases} \binom{m}{k} p^{2k}(1-p^2)^{m-k}, & \text{if } 0 \leq k \leq m \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

where k is the number of processes remaining in a group selected from m processes. A process will leave a group if, from the considered process' perspective, they do not respond to an "Are You Coordinator" message.

To invite other processes to the group, the two processes ultimately exchange up to 8 messages. In a round, a single process can invite many other processes to its group. From a selection of n other coordinators, the probability distribution for joining a new group with k of the n processes is:

$$\Pr_I(Y = k; n) = \begin{cases} \binom{n}{k} p^{8k}(1-p^8)^{n-k}, & \text{if } 0 \leq k \leq n \\ 0, & \text{otherwise.} \end{cases} \tag{6.9}$$

In the profile chain, in a state s that describes the number of processes in a group, the probability of transitioning from s to s' with n total processes (including the considered process) is:

$$\Pr_T(Z = s'; n; s) = \sum_{i=0}^{s-1} \Pr_M(X = i; s-1) * \Pr_I(X = s' - i; n - s - 1). \tag{6.10}$$

From this distribution, a set of transition probabilities can be calculated for a given probability of delivery p and number of processes n. This set of transition probabilities forms a profile Markov chain P, which can be evaluated for any number of processes n and probability of delivery p. The generated profile chain is ergodic when $0 < p < 1.0$. The profile chain is a stationary Markov chain.

6.4.4. Model Validation. To assert the closed-form profile chain accurately represents the implementation of the algorithm, it must be validated. Since T and P are ergodic, they can be checked for equivalence using a goodness-of-fit test. If the goodness-of-fit test indicates the chains are equivalent, they will generate similar sequences and have similar properties when analyzed. Therefore, generated P chains can be used to analyze the behavior of the algorithm during live execution with changing conditions.

To verify the test chain T is equivalent to the profile chain P, a χ^2 goodness-of-fit test is employed. The null-hypothesis of the test (H_0) asserts the profile chain P is equivalent to the test chain T:

$$H_0 : T = P. \tag{6.11}$$

With an alternative hypothesis that the two chains are not equivalent:

$$H_1 : T \neq P. \tag{6.12}$$

The χ^2 test measures the goodness-of-fit for a complete chain by combining the measurements of goodness of fit for the transitions away from each state. Therefore, the goodness of fit test for the chain is a summation of tests for each state:[10]

$$\chi^2 = \sum_{i}^{m} \sum_{j}^{m} = \frac{n_i(P_{ij} - T_{ij})^2}{P_{ij}} \tag{6.13}$$

where n_i is the number of times the state i was observed in the input sequence used to construct the test chain T. The summation is distributed as χ^2 with $m(m-1)$ degrees of freedom (DF) if all entries in P_{ij} are non-zero. In this work, all transitions in the profile Markov chain P are non-zero when $0 < p < 1.0$. However, the probability of some transitions may be extremely small. The χ^2 value was compared to a critical

Table 6.1: Summary of χ^2 tests performed.

Processes	DF	CV	Worst Error	Pr($WorstError$)	p-value
3	6	12.6	8.90	0.80	0.18
4	12	21.0	14.55	0.75	0.27
5	20	31.4	23.47	0.65	0.27
6	30	43.8	32.69	0.85	0.34

value (CV) giving a measure of how likely it was H_0 could not be rejected. This work selected an $\alpha = 0.05$ significance level to reject the hypothesis $T = P$.

If the hypothesis H_0 were to be rejected, it would indicate the test chain and profile chain differ significantly. As a consequence of rejecting the hypothesis, the implementation would have behavior from the generated closed-form solution.

To verify the model, it was compared to runs of an implementation of the algorithm. Test data was collected for systems with 3, 4, 5 and 6 processes. Information was collected from sufficiently long runs of the system with a delivery probability between 0.05 and 0.95 tested at intervals of 0.05. Table 6.1 shows the measured error and p-value for the worst observed error for each number of processes. Since the measured error is less than the critical value and the p-value is greater than 0.05, we cannot reject H_0. As a consequence, the profile chains (P) are representative of the behavior of the algorithm.

6.4.5. Profile Chain Analysis. Resources can only be managed effectively when multiple DGI coordinate together to manage those resources. Without another DGI to coordinate with, the DGI has a limited range of options to manage power generation, storage, and loads. Therefore, the amount of time the DGI will spend coordinating with another process is of particular interest. [38] defines an IGT metric to measure the amount of time a DGI process spends coordinating with at least one other process. In this work, we define IGT based on the steady state of the profile

chain. Let $\pi = Steady(P)$ for some profile chain. The IGT is the sum of all states in π, save the first state where the process is alone:

$$IGT = \sum_{i=2}^{m} \pi_i \tag{6.14}$$

The IGT is a number between 0 and 1. It represents the probability a random observation sees a group of at least two members. The steady state distributions are presented as stacked bar graphs in Figures 6.3, 6.4, 6.5 and 6.6. Each complete bar in the graph indicates the IGT. Additionally, these figures contain the average group size (AGS) when the system has reached the steady-state plotted as a fraction of the total number of processes. Let P be the steady-state distribution vector for some number of processes, n, and a given probability of delivery rate:

$$y = \frac{\sum_{i=1}^{n} P_i * i}{n} \tag{6.15}$$

where y is the plotted average group size (AGS) as a fraction.

The components of each bar represent the probability the system is in a specific state for a random observation of the system. The height of the component represents the relative probability of observing that state when in a group.

The profile chain can be used to ensure the FREEDM smart-grid is able to continue operating despite network issues. The profile chain can be combined with different message-sending strategies to maintain service. For example, the DGI can change its behavior to ensure operation continues normally despite connectivity issues. By selecting different strategies depending on the message delivery probability, the DGI can offer high performance in good network conditions and an acceptable level of service during faults. The profile chain can be extended to an arbitrary number of processes as shown in Figure 6.7. In the figure, the steady-state of the system is used to compute a weighted average of the group size. To compare the produced

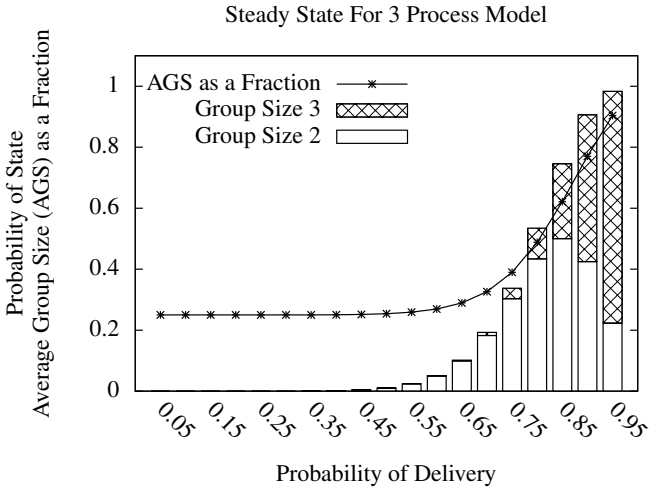

Figure 6.3: Steady state distribution for 3 processes as well as the AGS as a fraction of total processes.

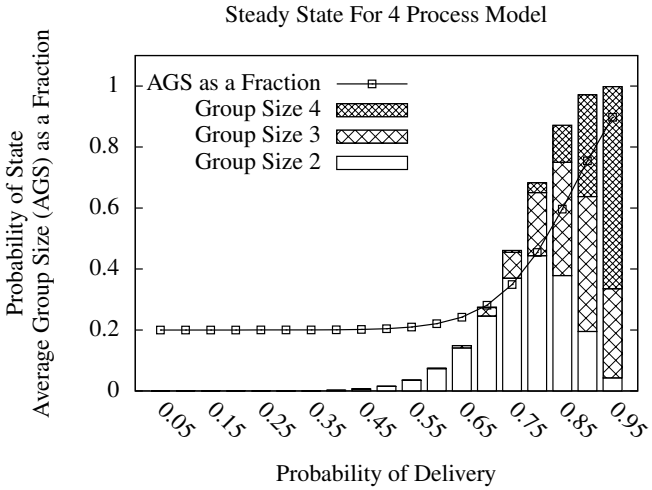

Figure 6.4: Steady state distribution for 4 processes as well as the AGS as a fraction of total processes.

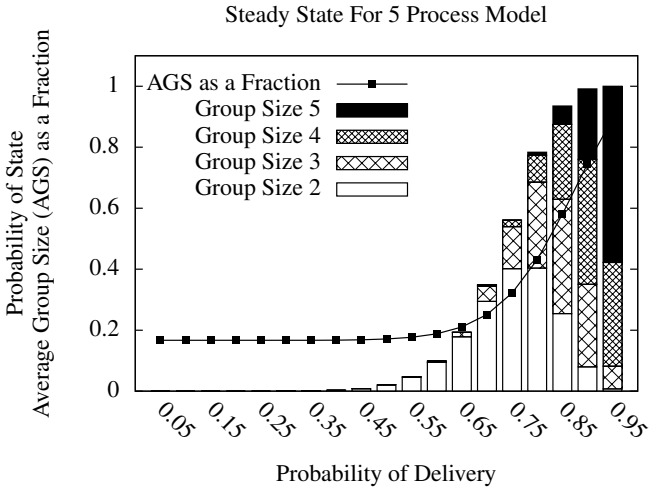

Figure 6.5: Steady state distribution for 5 processes as well as the AGS as a fraction of total processes.

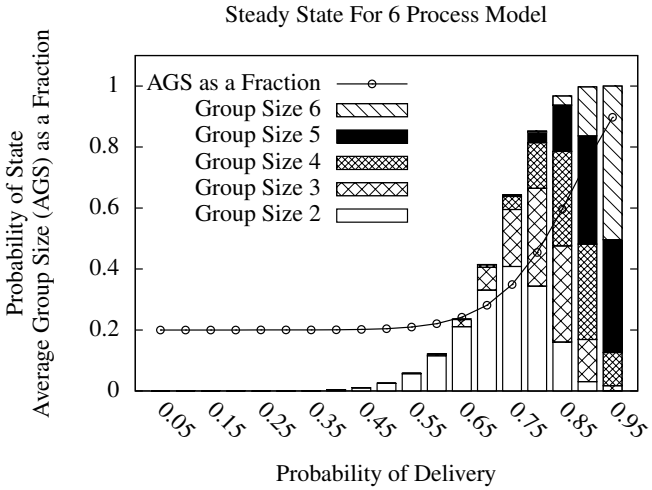

Figure 6.6: Steady state distribution for 6 processes as well as the AGS as a fraction of total processes.

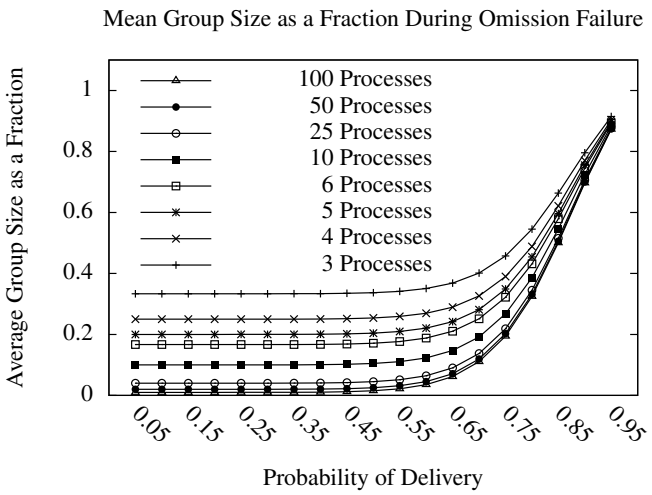

Figure 6.7: Average group size as a percentage of all processes in the system for larger systems.

steady states, the weighted average is plotted as a percentage of all processes in the system. Values in Figure 6.7 are plotted using Equation 6.15. Therefore, Figure 6.7 shows the average percentage of total processes that will be in the group in a steady-state system.

7. APPLICATION: ECN HARDENING

We considered the effects of network congestion on a cyber communication network used by the FREEDM smart-grid using a model of DGI processes in a partitioned network. By applying background traffic to the partitioned network, we create a situation where real-time deadlines are missed due to queuing delays. In the FREEDM smart-grid, the consequences of network congestion could result in several problematic scenarios. First, if the congestion prevents DGI from autonomously configuring using its group management system, processes cannot work together to manage power devices. Secondly, if messages arrive too late, or are lost, the DGI could apply settings to the attached power devices, driving the physical network to instability. Unstable settings could lead to problems in the power grid such as frequency instability, blackouts, and voltage collapse. Congestion response techniques allow the DGI to anticipate behavior during a fault and allow it to harden itself against the fault. In a smart-grid system, interfering traffic can originate from the use of public infrastructure[50][33], a misconfigured device, changing networking requirements as new devices enter the network, or from an attacker in a DoS attack.

The DGI employed a round-robin schedule where each module was given a predetermined amount of time to execute. The schedule was determined by the number of DGI that could be grouped together and the expected cyber network conditions. Additionally, the load balance module, which managed the power resources, was scheduled to run multiple times in a long block before the group was evaluated for failure. This schedule is depicted in Figure 7.1.

The system began by using group management to organize groups. Next, load balancing ran to manage power resources in the created group. Finally, state collection obtained a causally consistent state to be used for reporting and offline analysis.

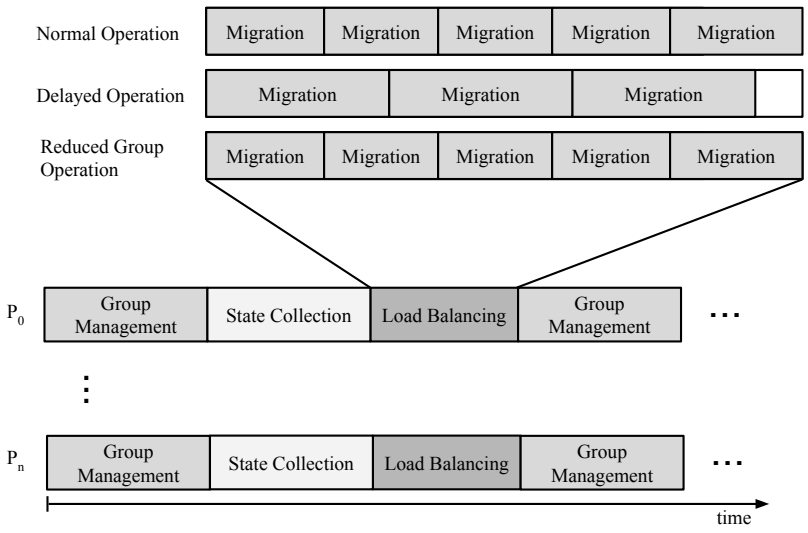

Figure 7.1: Example DGI schedule. Normal operation accounts for a fixed number of migrations each time the load balancing module runs. Message delays reduce the number of migrations that can be completed each round. However, reducing the group size allows more migrations to be completed (because fewer messages are being exchanged) at the cost of flexibility for how those migrations are completed.

For the purpose of the ECN experiments presented in the subsequent sections, we opted to not use the state collection module. If message delays occurred, the number of migrations load balancing could perform was reduced. However, by reducing the group size, the number of messages sent by load balancing could be reduced, allowing for a greater amount of work to be done. The outcome of an election had a direct effect on how effectively the DGI can manage resources.

This information can also be used to anticipate faults and mitigate them before they occur. Modern routers can supply an expected congestion notification as part of the IP header[23]. When congestion is anticipated, the coordinator can preemptively split the group to reduce congestion. This is possible because the load balancing algorithm's message complexity is $O(n^2)$. If future congestions causes an omission probability of as little as 0.15, performing a coordinated group division can potentially avoid transient states where processes cannot coordinate.

The coordinated group division has three main benefits: Breaking the large groups into smaller groups drastically reduces the number of messages transmitted and helps relieve congestion. The coordinator can design the split to ensure work continues by mixing supply and demand processes. The division can also account for the placement of routers for targeted congestion relief.

7.1. THEORY OF OPERATION

In the simulated network, the router and the two switches were ECN enabled devices. The devices monitored their incoming packet queues for each of their interfaces and used the RED algorithm to determine when to mark a packet for ECN. Since ECN fields in an IPv4 header are not directly available to an application, the notifications were multicast onto the source interface(s) of the packet that triggered

the notification. Each network device ran an application responsible for generating the multicast ECN message.

When the RED algorithm identifies congestion, it must notify senders. Because the approach is non-standard and most UDP applications would not understand the notification, we opted to create an application to run on switches and routers. When congestion is detected, the application sends a multicast beacon to a group of interfaces informing the attached devices of the level of congestion. For similarity with the RED algorithm and the NS-3 implementation, the notification is classified as either "soft" or "hard." A soft notification is an indication the congestion in the network is approaching a level where real-time processes can expect message delays which may affect their normal operation. A hard notification indicates the congestion has reached a level where messages are subject to both delay and loss.

7.2. GROUP MANAGEMENT

The group management module's execution schedule is broken into several periods of message generation and response windows. Because the schedule of the DGI is partially synchronous, the traffic generated by modules occurs in bursts. The number of messages sent by group management is $O(n^2)$ (where n is the number of processes in the system). The duration of the response window is dependent on the amount of time it takes for messages to propagate to the hardest-to-reach process. To contend with congestion, slack, in the form of idle time, must be added to allow the RED algorithm to detect congestion before it reaches a critical level. Figure 7.2 depicts typical queuing behavior for a network device serving DGI processes under different circumstances.

Because the traffic generated by DGI modules occurs in bursts, there is a phenomenon where the traffic bursts mix with steady background traffic to cause

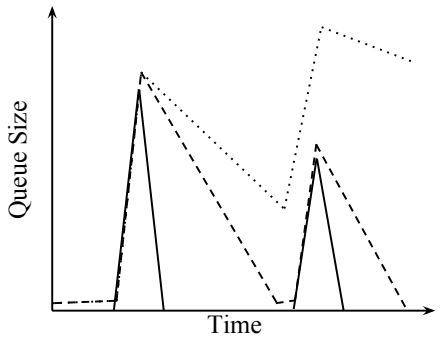

Figure 7.2: Example of network queuing during DGI operation.
DGI modules are semi-synchronous and create traffic bursts on the network. When there is no other traffic on the network (solid line), the traffic bursts cause many packets to queue quickly, but the queue empties at a similar rate. With background traffic (dashed line), the traffic bursts cause many packets to be queued suddenly. More packets arrive continuously, causing the queue to drain off more slowly. When the background traffic reaches a certain threshold (dotted line), the queue does not empty before the next burst occurs. When this happens, messages will not be delivered in time, and the queue may completely fill.

the packet queue at the network device to fill. With no background traffic, the impulse queues many messages, but those messages are distributed within real-time constraints. When the background traffic is introduced, the queue takes longer to empty. At a critical threshold, the queue does not empty completely before the next burst is generated by the DGI and services provided by the DGI are disrupted. If the level of other traffic is sufficiently high, the queue completely fills, and no messages can be distributed. The RED algorithm and ECN are used to delay or prevent the queue from reaching this critical threshold.

7.2.1. Soft ECN. A soft ECN message indicates the network has reached a level of congestion where the router suspects processes will not be able to meet their real-time requirements. The soft ECN message encourages the DGI processes to decrease the number of messages they send, reducing the amount of congestion they contribute to the network and to allow for reliable distribution techniques to have additional time to deliver messages (since fewer messages are being sent). In

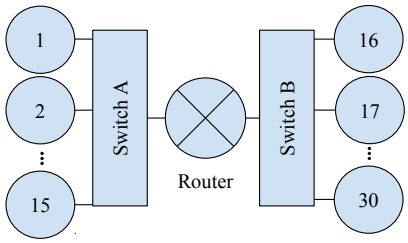

Figure 7.3: Example of process organization. Two groups of processes are connected by a router.

the case of potential congestion, the group management module can reduce its traffic bursts by disabling elections during the congestion. When elections are disabled, messages for group management are only sent to members of the group. Processes do not seek out better or other leaders with which to merge. As a consequence, the message complexity for processes responding to the congestion notification reduces from $O(n^2)$ to $O(n)$.

7.2.2. Hard ECN. In a hard ECN scenario, the router will determine congestion has reached a threshold where the real-time processes will soon not be able to meet their deadlines. In this scenario, the real-time process will likely split its group. In an uncontrolled situation, the split will be random, and time will be wasted while the system is reorganizing. It is therefore desirable when this level of traffic is reached to split the group. Splitting the group reduces the number of messages sent across the router for modules with $O(n^2)$ (where n is the number of processes in the original group) message complexity. For larger groups, splitting them provides significant savings in the number of messages that must be queued by the router, especially since the traffic occurs in bursts.

Consider a network like one depicted in Figure 7.3, where processes are divided by a router. Here, there are n processes on one side of the network and m on the other. In normal operation, the omission-modelable algorithm has an $O(n^2)$ message

complexity. In Soft ECN maintenance mode, the reduced number of messages reduces the complexity to $O(n)$ by disabling elections.

During elections (and with each group update) the leader distributes a fallback configuration that will coordinate the division of the groups during intense congestion. When the ECN notification is received, the processes will halt all current group management operations and enter a splitting mode where they switch to the fallback configuration. The leader of the group distributes a fallback notification to ensure all processes in the group apply their new configuration. The complexity of distributing the notification is linear $O(n)$ and processes that already received the notification will have halted their communication. This approach will ideally avoid the burst/drain phenomena from Figure 7.2.

The design of the fallback configuration can be created to optimize various factors. The factors include cyber considerations, such as the likely network path the processes in the group will use to communicate. By partitioning the group around cyber network resources, the group can be selected to minimize the amount of traffic that crosses the congested links in the future. Additionally, constraints of the physical network can be considered. Fallback groups can be created to ensure they can continue to facilitate the needs of the members. The design of the configuration can take into the consideration the distribution of supply and demand processes in the current group. By having a good mix of process types in the fallback group, the potential for work can remain high.

7.3. CYBER-PHYSICAL SYSTEM

For a real-time CPS, message delays could affect coordinated actions. As a result, actions may not happen at the correct moments, or at all. Since the two-army problem prevents any process from being entirely certain a coordinated action will

happen in concert, problems arising from delay or omission of messages is of particular interest. Specifically, we are interested in the scenario from [16], where only half of a power migration is performed. Other power management algorithms could have similar effects on the power system based on this idea of a process performing an action that does not have a corresponding, compensating action.

7.3.1. Soft ECN Notification. In a soft congestion mode, the process being informed of the congestion can reduce its effect on the congestion by changing how often it generates traffic bursts. Processes running the load balancing algorithm make several traffic bursts when they exchange state information and prepare migrations. As shown before, if the interval between these bursts is not sufficient for the queue to drain before the next burst occurs, then critical, overwhelming congestion occurs. The schedule of the DGI is fixed at run-time and processes cannot simply extend the duration of the load balancing execution phase. However, on notification from the leader, the process can reduce the number of migrations to increase the message delivery interval. The notification to reduce the schedule originates from the coordinator as part of the message exchange necessary for the process to remain in the group. Every process in the group must receive the message to participate in load balancing, ensuring all processes remain on the same real-time schedule. By using this approach, the amount of traffic generated is unchanged but the period a process waits for the messages to be distributed is increased.

7.3.2. Hard ECN Notification. When the DGI process receives a hard congestion notification, the processes switch to a predetermined fallback configuration. The configuration creates a cyber partition. By partitioning the network, the number of messages sent by applications with $O(n^2)$ message complexity can be reduced significantly. Each migration of the load balancing algorithm begins with an $O(n^2)$ message burst and so benefits from the reduced group size created by the partition.

Consider a network like the in Figure 7.3 with n processes on one half and m on the other. The number of messages sent across the router for the undivided group is of the order $2mn$ as the n processes on side A send a message to the m on side B and vice-versa. Let i_1 and j_1 be the number of processes from side A and side B (respectively) in the first group created by the partition. Let i_2 and j_2 be the number of processes in the second group created by the partition under the same circumstances of i_1 and j_1. The number of messages sent that pass through the router, is then:

$$2i_1j_1 + 2i_2j_2. \tag{7.1}$$

For an arbitrary group division, the following can be observed. Suppose i_1 and j_2 are the cardinality of two arbitrarily chosen sets of processes from side A and side B respectively. Following the same cut requirements as before:

$$i_2 = n - i_1 \text{ and } j_2 = m - j_1. \tag{7.2}$$

The number of messages that must pass through the router for this cut is:

$$2i_1j_1 + 2(n - i_1)(m - j_1). \tag{7.3}$$

The benefits of the cut are minimized when i_1 and j_1 are $\frac{n}{2}$ and $\frac{m}{2}$:

$$2(2\frac{mn}{4} + mn - \frac{mn}{2} - \frac{mn}{2}) = mn \tag{7.4}$$

which is a reduction of half as many messages. For systems with many participating processes, a significant reduction in the number of messages sent across the router is achieved. As a consequence, this further extends the delivery window for processes sending messages by decreasing channel utilization.

7.4. RELATION TO OMISSION MODEL

The synchronization of clocks in the environment is assumed to be normally distributed around a true time value provided by the simulation. The shape of the curve created by plotting the queue is a Cumulative Distribution Function (CDF) of the normal distribution, noted as $F(x)$. A simple description of the traffic behavior can then be described in terms of that curve. First, when the queue hits a specific threshold, even if the queue is drained at an optimal rate, the nth queued packed will not be delivered in time:

$$Qsize - min(Qsize, (DequeueRate * \Delta t)) \geq 0 \tag{7.5}$$

where Δt is the deadline for the message to be delivered. If the size of the queue exceeds the number of messages that can be delivered before Δt passes, some messages will not be delivered. The size of the queue during the message bursts created by the DGI depends on the message complexity of the algorithm, the number of messages already in the queue, the other traffic on the network, and any replies needing to be delivered in that interval. Therefore, let c represent the rate traffic is generated by other processes. Let $init_q$ represent the number of messages in the queue at the start of a burst. Let $init_m$ represent the number of messages sent in the beginning of the burst. Let $resp$ represent the number of messages sent in response to the burst that must still be delivered before Δt passes. We can then express $Qsize$ as two parts:

$$Qsize = Burst + Obligations \tag{7.6}$$

where $Burst$ takes the form of the CDF for the normal distribution:

$$Burst = init_m * F(x) \tag{7.7}$$

$$Obligations = c * \Delta t + init_q + resp. \tag{7.8}$$

From this we can derive the equation:

$$F(x) \geq \frac{DequeueRate * \Delta t - c * \Delta t - init_q - resp}{init_m} \tag{7.9}$$

where, from Equation 7.5, $DequeueRate * \Delta t$ is less than or equal to the number of messages in the queue. Solving for $F(x)$ gives a worst case estimate of the probability of delivery for a specific algorithmic or network circumstance. *DequeueRate* is affected by the amount of traffic in the system. It should be noted, a greater amount of background traffic corresponds to a greater average queue size. From a relationship between the background traffic, the average queue size, and the results presented in the previous chapters, Equation 7.9 can be used to select the ECN parameters.

7.5. PROOF OF CONCEPT

7.5.1. Experimental Setup. Experiments were run in a Network Simulator 3.23[18] test environment. The simulation time replaced the wall clock time in the DGI for the purpose of triggering real-time events. As a result, the computation time on the DGI for processing and preparing messages was neglected. However, to compensate for the lack of processing time, the synchronization of the DGI was randomly distributed as a normal distribution. This was done to introduce realism to ensure events did not occur simultaneously. Additionally, the real-time schedules used by the DGI were adjusted to remove the processing time that was neglected in the simulation. The schedule used is depicted in Figure 7.4.

The DGI were placed into a partitioned environment. The test included 30 nodes. Each of the nodes ran one DGI process. Two sets of 15 DGI were each connected to a switch and each switch was in turn connected to the router. The

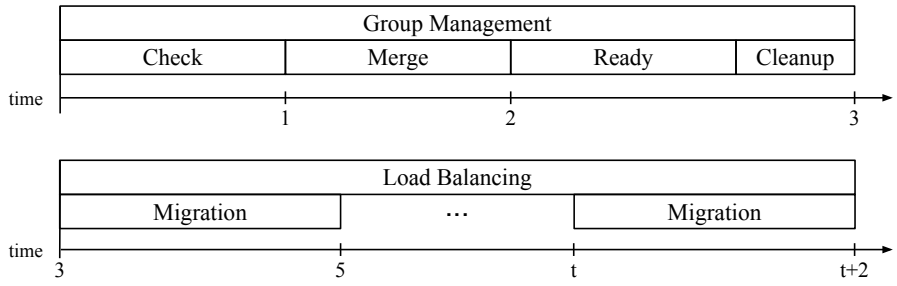

Figure 7.4: Execution schedule used in experiments. Each burst of messages was allotted one second for those messages and their replies to propagate. During the load balancing execution period, ten migrations are performed on the normal schedule. Soft ECN increased the message propagation time for the load balancing module to 1.5 seconds and reduced the number of migrations each round to six.

network is pictured in Figure 7.3. Node identifiers were randomly assigned to nodes in the simulation and used as the process identifier for the DGI. In the experiments, the load at each process was randomly assigned: each side is mix of supply and demand processes.

The links between the router and the switches had a RED enabled queue placed on both network interfaces. The RED parameters for all queues were set identically. A summary of RED parameters are listed in Table **??**. All links in the simulation were 100Mbps links with a 0.5ms delay. RED was used in packet count mode to determine congestion. ARP tables were populated before the simulation began. RED parameters were selected using results from [39].

The relationship between the background traffic and the average queue size was estimated through runs of the NS-3 simulation. Figure 7.5 demonstrates the observed relationship between the total background traffic and the maximum average queue size for that level of traffic. Additionally, the *DequeueRate* was collected from a run of the simulation without traffic, and was found to be 713.08 packets per second. Therefore, from Equation 7.9, assuming $init_q = 0, resp = 225, init_m = 225$, and $\Delta t = 1$, the maximum traffic rate with no omissions was 263.0 packets per

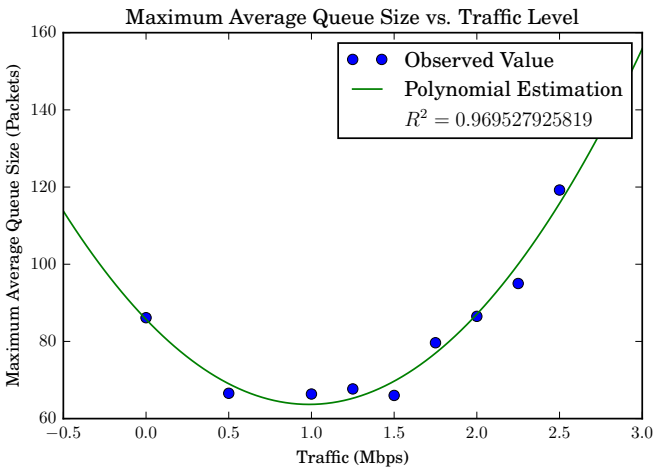

Figure 7.5: Plot of the maximum observed average queue size as a function of the overall background traffic. The polynomial estimate is $y = 22.70x^2 - 44.74x + 85.72$

Table 7.1: Summary of RED parameters.

Parameter	Value	Parameter	Value
RED Queueing Mode	Packet	RED Gentle Mode	True
RED Q_w	0.002	RED Wait Mode	True
RED Min Threshold	90	RED Max Threshold	130
RED Link Speed	100Mbps	RED Link Delay	0.5 ms

Unspecified values default to the NS-3 implementation default value

second. The number of packets for the *resp* and $init_m$ were selected from the worst case of the modified election algorithm. Based on the traffic parameters in Table **??**, 263.0 packets/second corresponded to 1.077Mbps of traffic generated at one switch and 2.1545Mbps traffic overall. From the polynomial estimate in Figure 7.5, the maximum average queue size for that level of traffic was 94.715, estimated as 90 for the RED Min Threshold in Table **??**. RED Max Threshold is computed using a similar technique, but using the message complexity for the load balancing algorithm, since it maintained its complexity during Soft ECN mode.

To introduce traffic, processes attached to each of the switches attempted to send a high volume of messages to each other across the router. The number of

Table 7.2: Summary of Test Configurations

Test	Traffic	Notifications	Attempted Migrations
A	None	N/A	1171
B	2Mbps	N/A	1171
C	4Mbps	None	1114
D	4Mbps	Soft	888
E	6.4Mbps	Soft	861
F	6.4Mbps	All	888

packets sent per second was a function of the data rate and the size of the packets sent. In each simulation, half of the traffic originated from each switch. Due to the properties of the network links, the greatest queueing effect occurred at the switches.

7.5.2. Results. Six test scenarios were created to evaluate the ECN technique. These scenarios are summarized in Table 7.2. In each scenario, the distribution of supply and demand processes is mixed: each switch has both supply and demand processes. The potential of the system to complete migrations, as well as the potential for migrations to fail were primary considerations for each scenario. Processes cannot initiate migrations if they are not grouped with other processes. Similarly, if the process is attempting to communicate over a congested link, it may not meet real-time deadlines or successfully complete migrations.

Figures 7.7 and 7.8 (Scenario A) show the normal operation of the system. In this configuration, there is no congestion on the network. The DGI start, group together and then begin migrating power between processes. Figure 7.7 plots the queue size over time for a queue used to send packets from a switch to the router. Figure 7.8 is a detailed view of a portion of Figure 7.7; it shows the queue size during the normal operation of group management as well as the first three migrations of a load balancing round. The dotted line plots the EWMA of the size of the queue. An important note for the plotted EWMA values: NS-3 only updates the average queue size during simulation when a new packet is enqueued. As a consequence,

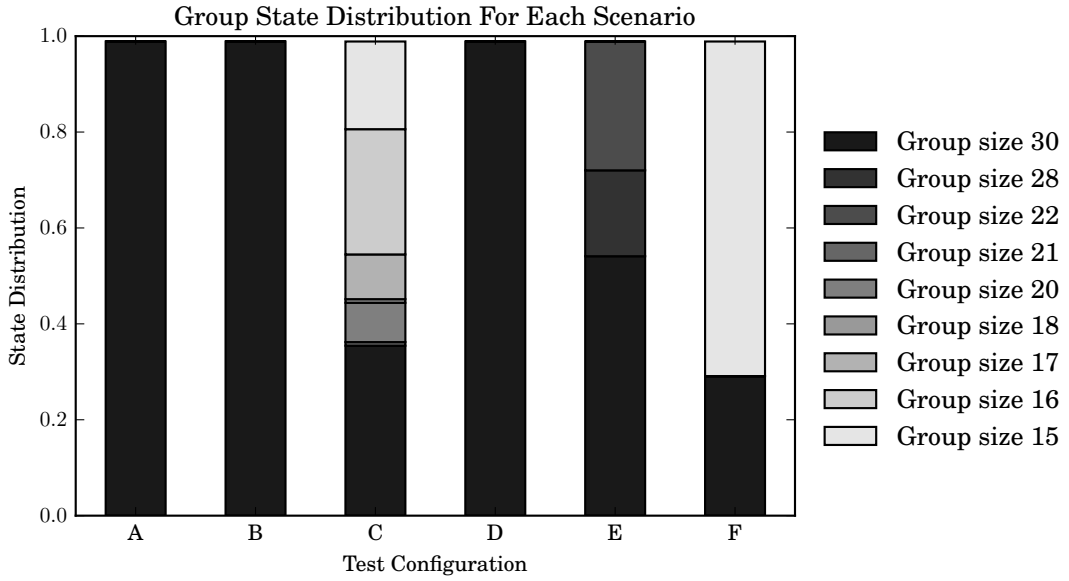

Figure 7.6: Distribution of group sizes for process 0 in each scenario. The list of scenarios is presented in 7.2. The height of each colored segment indicates the portion of the simulation where process 0 was in the given configuration.

the average queue size is slightly misleading in scenarios where there is a significant amount of idle channel time (such as Figures 7.7, 7.8, and 7.9). In scenarios A and B, the traffic is not sufficient to disrupt the operation of the DGI. As a result, in Figure 7.6, process 0 has an uninterrupted observation of a group size of 30 throughout the two scenarios.

From scenario A and B we establish the min_{th} value used as a RED queue parameter. The traffic generated by each step of the group management algorithm occurs in bursts. The tightness of the clock synchronization in the group affects the size of the peak. Like [65], the level of power at a process is the net sum of its power generation capability and load. As power is shared on the network, processes with excess generation, converge toward zero net power. Demand processes also converge toward zero net power.

Figure 7.9 shows the queue size as the network traffic begins to increase. The DGI in these experiments used a schedule that allowed for some congestion to occur

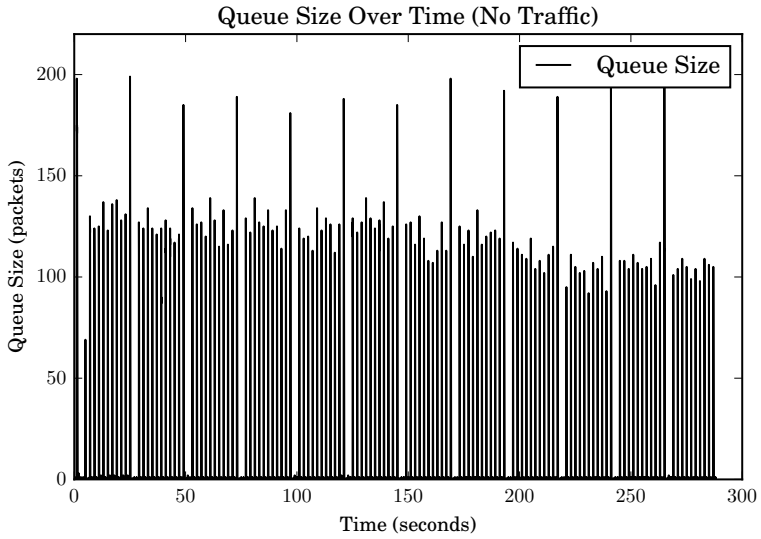

Figure 7.7: Plot of the queue size for a queue from switch A to the router when only the DGI generates traffic.
[Test configuration A. Plot of the queue size for a queue from switch A to the router when only the DGI generates traffic.]

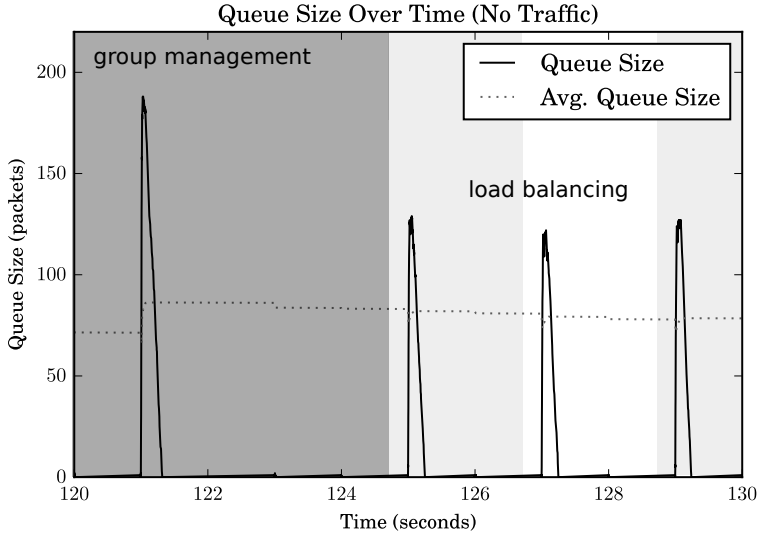

Figure 7.8: Test configuration A. Detailed view of Figure 7.7. In the figure, the dark gray section is the period where the group management module executes. The alternating white and light gray sections are the execution periods for individual migrations. In the normal schedule, the DGI allocates approximately 1 second to each migration and performs 10 migrations each round. Three migrations are shown.

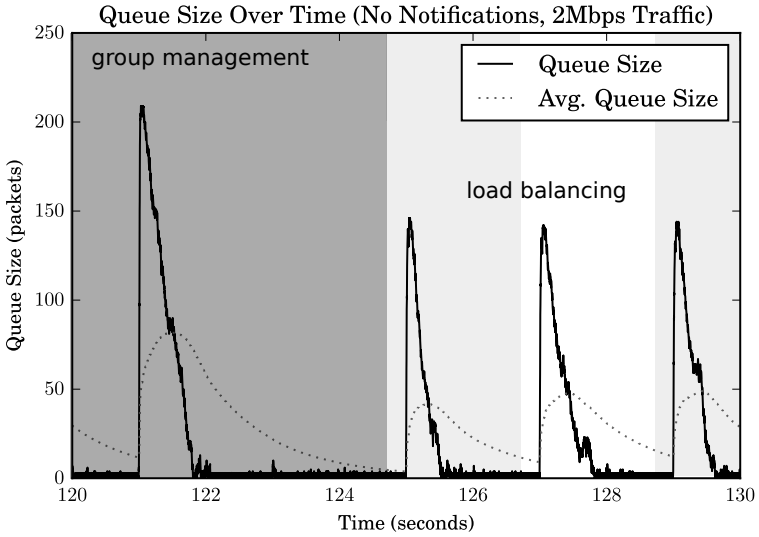

Figure 7.9: Test configuration B. Detailed view of the effect on queue size as other network traffic is introduced. Compared to Figure 7.8, the peaks are taller and wider. The background traffic creates the intermediate effect described in 7.2. As with Figure 7.8, this figure shows three migrations.

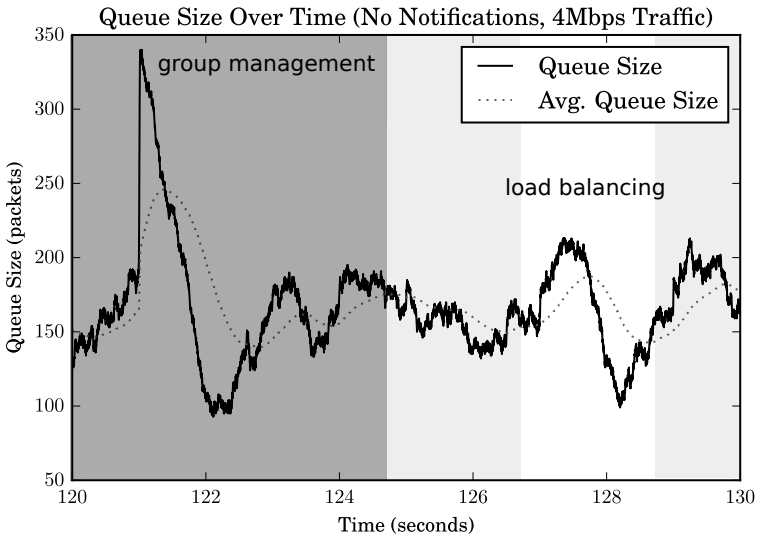

Figure 7.10: Test configuration C. Detailed view of the effect on queue size as other network traffic is introduced without notifications. In this scenario, the queue does not empty completely before the next burst of traffic occurs. As a result, group management generates a large peak as it tries to rejoin groups that have broken due to the message delays. Load balancing has smaller and less pronounced peaks because the groups are smaller. Additionally, during this scenario some migrations are lost due to message delays.

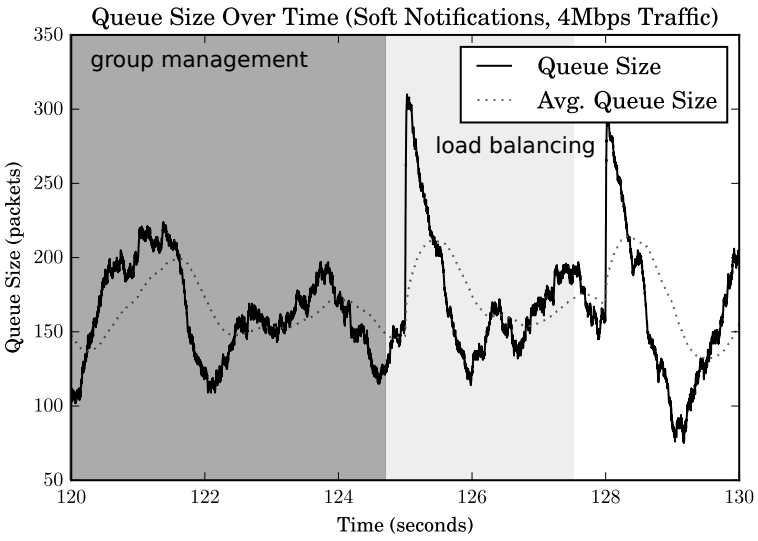

Figure 7.11: Test configuration D. Detailed view of the effect on queue size as other network traffic is introduced with soft ECN notifications. In this scenario, the group management module entered a maintenance mode where it suspended discovering leaders of other groups. As a result, that module's message complexity dropped significantly, resulting in the lack of a noticeable peak compared to previous scenarios. Additionally, the system decreased the number of migrations performed per round by load balancing. As a consequence, only two migrations are pictured. Load balancing has higher peaks than in Figure 7.10 because groups have not been disbanded. However, the reduced schedule is sufficient to prevent any migrations from being lost.

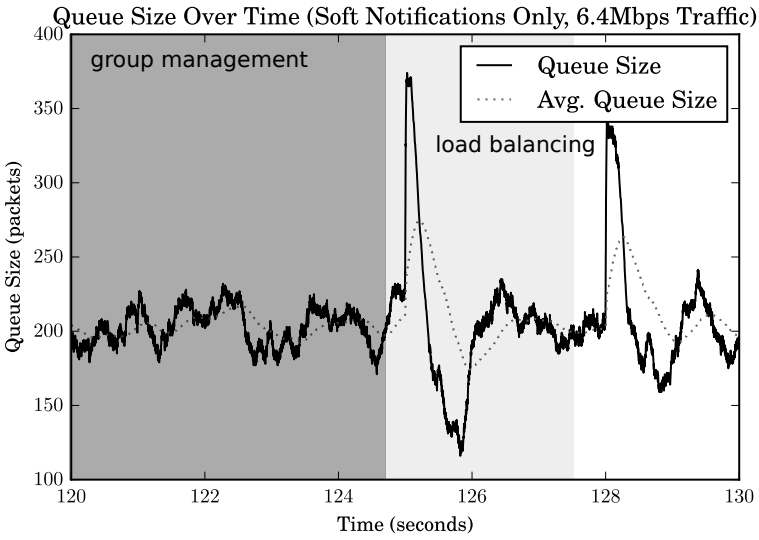

Figure 7.12: Test configuration E. Detailed view of the effect on queue size as a large amount network traffic is introduced. In this scenario, group management enters a maintenance mode when it receives the soft ECN notification, but that approach alone is not sufficient to maintain groups; processes leave the group randomly during the simulation. The system decreased the number of migrations per round, as it had done in Figure 7.11. However, migrations were lost.

before processes are disrupted. The slack gave network devices the opportunity to identify when the network congestion would go beyond the acceptable threshold.

Figure 7.10 (Scenario C) shows an example of congestion affecting the physical network without ECN. As a result of the congestion in Figure 7.10, processes leave the main group, shown in 7.6. The observed configurations have greater than 15 processes because communication that does not cross the switch-router-switch path is not congested. As a consequence, processes on the same switch as process 0 do not leave 0's group. Power migrations are affected by congestion: migrations fail, or the supply process is left uncertain of a migration's completion. Figure 7.14 plots the count of failed migrations over time. The number of failed migrations relative to the number attempted is quite low, because processes on the same switch as process 0 can still attempt migrations.

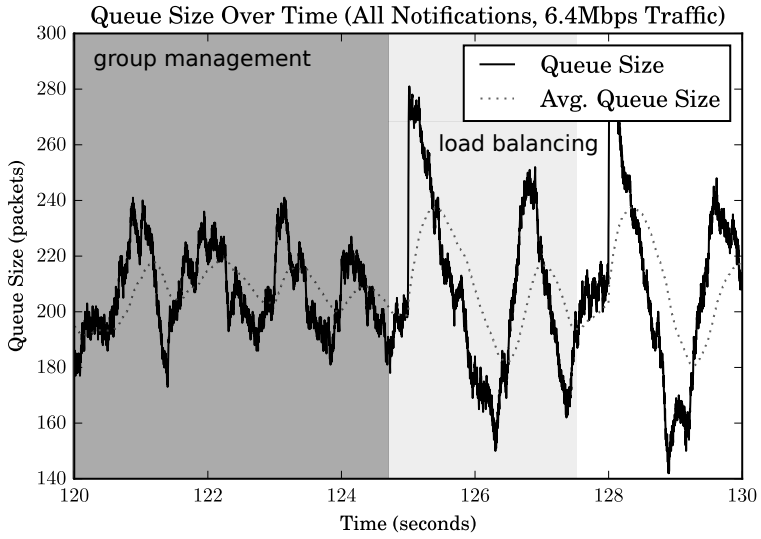

Figure 7.13: Test configuration F. In this scenario, hard notifications caused the groups to perform a coordinated division, enabled maintenance mode for the group management modules, and reduced the migration schedule for the load balancing module. As a consequence, the group management module produced more traffic than it did in Figure 7.12, but, the installed fallback configurations did not have any of the DGI leave the group. Additionally, the smaller group size allowed the amount of traffic from the load balancing module to be reduced while still allowing the module to migrate power between DGI without any migrations being lost.

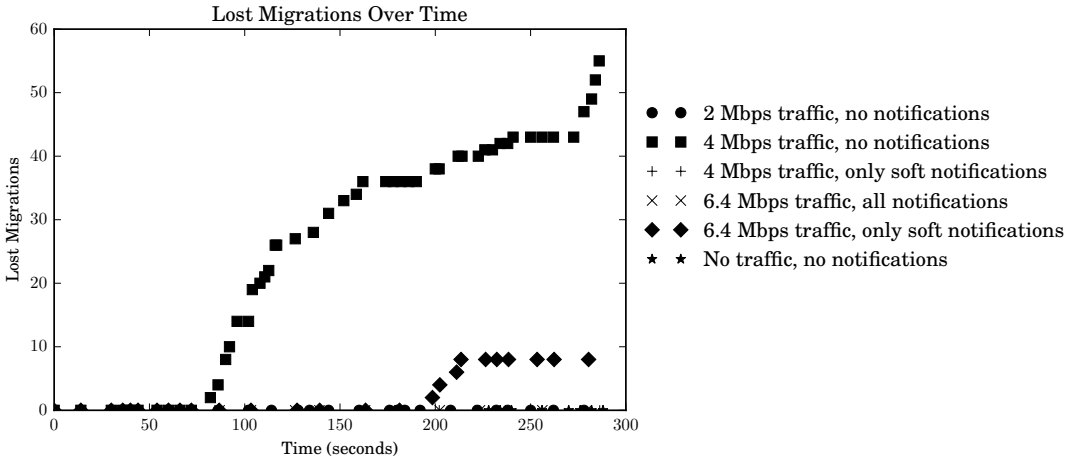

Figure 7.14: Count of lost migrations from all processes over time. Migrations are counted as lost until the second process confirms it has been completed. Without congestion management, migrations are lost.

Figure 7.11 (Scenario D) shows an example of the ECN algorithm notifying processes of the congestion. Compared to the scenario in Figure 7.10 (Scenario C), the ECN algorithm successfully prevents the group from dividing. As part of the compensation for the congestion, the number of migrations attempted are reduced, as listed in Table 7.2. In [15], when the number of migrations are reduced, the size of each migration is increased. Despite fewer migrations being attempted, the same amount of power can be managed by the DGI, and the number lost migrations is reduced by the changed schedule.

Figures 7.12 (Scenario E) and 7.13 (Scenario F) show an example of a more extreme congestion scenario. In Figure 7.13, the RED algorithm shares a Hard ECN notification. The notification causes the DGI to switch to a smaller fallback configuration, which decreases the queue usage from Figure 7.12 to Figure 7.13. Without this fallback configuration behavior, the system is greatly affected by the traffic. However, with the fallback configuration, the system remains stable and no migrations are lost.

8. CONCLUSION

We presented a useful framework for reasoning about distributed systems that tolerate omission faults. The models and structures presented in the framework allow algorithms to be designed with behaviors that can be modeled with a Markov chain.

To create the framework, existing information flow analysis techniques were applied to common distributed systems problems. We showed the information flow-based analysis was consistent with previous work. The analysis was extended to reason about a leader election algorithm. We described belief sets created by distributing information to several other agents in the system and showed which portions were MSDND secure.

Additionally, we defined how information being transferred between agents and the actions they take based on that information could be modified to have the memorylessness property of Markov chains. Using this concept we demonstrated how a common leader election algorithm could be modified to use this memorylessness property, allowing it to be modeled online during changing conditions.

Our work is particularly valuable for the analysis of critical infrastructure systems, where knowledge of their behavior during fault conditions is important. By allowing the ability for algorithms to determine what issues are likely to arise while they are operating, actions can be taken to protect the infrastructure from failure. There are a wide range of possible applications, including actions either undertaken by human operators on site, or autonomous actions taken by the algorithms to harden themselves against failure. We presented a technique for hardening a real-time, distributed cyber-physical system against network congestion. The RED queueing algorithm and an out-of-band version of explicit congestion notification (ECN) were used to signal an application of congestion. Using this technique, the application changed

several of its characteristics to ready itself for the increased message delays caused by the congestion.

Techniques were demonstrated on the DGI, a distributed control system for the FREEDM smart-grid project. We showed the hardening techniques were effective in keeping the DGI processes grouped together. Additionally, the changes applied to the DGI through cyber-coordinated actions helped prevent potential destabilization of the physical power network. Our techniques could be applied to any CPS that could experience congestion on its network, as long as it has the flexibility to change its operating mode. Potential applications can apply to both the cyber control network and the physically controlled process.

APPENDIX

ALGORITHM

The modified leader election algorithm.

1: $AllPeers \leftarrow \{1, 2, ..., N\}$

2: $Coordinators \leftarrow \emptyset$

3: $UpPeers \leftarrow Me$

4: $State \leftarrow Normal$

5: $Coordinator \leftarrow Me$

6: $Expected \leftarrow \emptyset$

7: $Counter \leftarrow$ A random initial identifier

8: $GroupID \leftarrow (Me, Counter)$

9: $PendingID \leftarrow (Me, -1)$

10: $Pending \leftarrow \emptyset$

11:

12: **function** CHECK

13: This function is called at the start of a round by a leader

14: **if** $State \neq Normal$ or $Coordinator \neq Me$ **then return**

15: **end if**

16: $Expected \leftarrow \emptyset$

17: **for** $j \in (AllPeers - \{Me\})$ **do**

18: $AreYouCoordinator(j)$

19: $Expected \leftarrow Expected \cup j$

20: **end for**

21: Peers which respond "Yes" to $AreYouCoordinator$ are put into the $Coordinators$ set.

22: Processes that respond are removed from $Expected$.

23: When an *AreYouThere* response is "No" and this process is a coordinator, the querying process is put in the *Coordinators* set.

24: Wait for responses, Peers that do not respond are removed from UpPeers.

25: $UpPeers \leftarrow (UpPeers - Expected) \cup Me$

26: $UpPeers \leftarrow (UpPeers - Coordinators) \cup Me$

27: **if** $Responses = \emptyset$ **then return**

28: **end if**

29: **if** $Me = SelectedProcess$ **then**

30: MERGE(Responses)

31: **end if**

32: **end function**

33:

34: **function** TIMEOUT

35: This function is called at the start of a round by the group members

36: **if** $Coordinator = Me$ **then return**

37: **else**ARE YOU THERE(Coordinator,GroupID,Me)

38: **if** Response is No **then** RECOVERY

39: $Coordinators \leftarrow Coordinator$

40: **end if**

41: **end if**

42: **end function**

43:

44: **function** MERGE(Coordinators)

45: This function invites all coordinators in Coordinators to join a group led by Me

46: $State \leftarrow Election$

47: Stop work

48: $Counter \leftarrow Counter + 1$

49: $PendingID \leftarrow (Me, Counter)$

50: $Coordinator \leftarrow Me$

```
51:     Pending ← UpPeers − Me
52:     for j ∈ Coordinators do INVITE(j,Coordinator,PendingID)
53:     end for
54:     Wait for responses, Peers that accept the invite are added to Pending.
55:     State ← Reorganization
56:     GroupID ← PendingID
57:     UpPeers ← Pending
58:     for j ∈ UpPeers do READY(j,Coordinator,GroupID,UpPeers)
59:     end for
60:     Expected ← UpPeers
61:     Wait for responses, Peers that acknowledge are removed from Expected
62:     UpPeers ← UpPeers − Expected
63:     State ← Normal
64: end function
65:
66: function RECEIVEREADY(Sender, Leader, Identifier, Peers)
67:     if State = Reorganization and PendingID = Identifier then
68:         UpPeers ← Peers
69:         State ← Normal
70:         Coordinator ← Leader
71:         GroupID ← Identifier READYACKNOWLEDGE(Leader,Identifier)
72:     end if
73: end function
74:
75: function RECEIVEAREYOUCOORDINATOR(Sender)
76:     if State = Normal and Coordinator = Me then
77:         Respond Yes
78:     else
79:         Respond No
```

```
80:     end if
81: end function
82:
83: function ReceiveAreYouThere(Sender, Identifier)
84:     if GroupID = Identifier and Coordinator = Me and Sender ∈ UpPeers then
85:         Respond Yes
86:     else
87:         Respond No
88:         Coordinators ← Sender
89:     end if
90: end function
91:
92: function ReceiveInvite(Sender,Leader,Identifier)
93:     if State ≠ Normal then return
94:     end if
95:     if Sender ≠ SelectedProcess then return
96:     end if
97:     Stop Work
98:     PendingID ← Identifier
99:     State ← Election
100:    Accept(Coordinator, Identifier)
101:    State ← Reorganization
102:    if Ready is not received then
103:        Recovery()
104:    end if
105: end function
106:
107: function ReceiveAccept(Sender,Leader,Identifier)
108:    if State ← Election and PendingID = Identifier then
```

```
109:          Pending ← Pending ∪ Sender
110:      end if
111: end function
112: function ReceiveReadyAcknowledge(Sender)
113:      Pending ← Pending ∪ Sender
114: end function
115:
116: function Recovery
117:      State ← Election
118:      Stop Work
119:      Counter ← Counter + 1
120:      GroupID ← (Me, Counter)
121:      Coordinator ← Me
122:      UpPeers ← Me
123:      State ← Reorganization
124:      State ← Normal
125: end function
```

BIBLIOGRAPHY

[1] Communication requirements of smart grid technologies. Technical report, U.S. Department of Energy, Oct 2010.

[2] IEEE standard for electric power systems communications-distributed network protocol (DNP3). *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pages 1–821, Oct 2012.

[3] R. Akella, Fanjun Meng, D. Ditch, B. McMillin, and M. Crow. Distributed power balancing for the FREEDM system. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 7 –12, October 2010.

[4] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The spread toolkit: Architecture and performance. Technical report, Johns Hopkins University, 2004.

[5] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: a communication subsystem for high availability. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 76–84, 1992.

[6] Chongyang Bai and Xuejun Zhang. Aircraft landing scheduling in the small aircraft transportation system. In *Computational and Information Sciences (ICCIS), 2011 International Conference on*, pages 1019–1022, Oct 2011.

[7] Zubair A. Baig and Abdul-Raoof Amoudi. An analysis of smart grid attacks and countermeasures. *Journal of Communications*, 8(8):473–479, Aug 2013.

[8] J. Baillieul and P. J. Antsaklis. Control and communication challenges in networked real-time systems. *Proceedings of the IEEE*, 95(1):9–28, Jan 2007.

[9] F. Baker. Requirements for IP version 4 routers, 6 1995. RFC 1812.

[10] U. Narayan Bhat. *Elements of applied stochastic processes, 2nd Edition*. John Wiley & Sons, 1984.

[11] K. Birman and Renesse R. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA 90720-1264, 1994.

[12] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. In *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '92, pages 147–158, New York, NY, USA, 1992. ACM.

[13] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996.

[14] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[15] A. Choudhari, H. Ramaprasad, S. Chellappan, B. McMillin, J. Kimball, and M. Zawodniok. Adaptive scheduling with explicit congestion notification in a cyber-physical smart grid system. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 309–317, Aug 2014.

[16] A. Choudhari, H. Ramaprasad, T. Paul, J.W. Kimball, M. Zawodniok, B. McMillin, and S. Chellappan. Stability of a cyber-physical smart grid system using cooperating invariants. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 760–769, July 2013.

[17] OMNeT++ Community. Omnet++, May 2012. `http://http://www.omnetpp.org/`.

[18] NS-3 Consortium. Network simulator 3.23. `http://www.nsnam.org/`.

[19] Flaviu Cristian. Understanding fault-tolerant distributed systems. *COMMUNICATIONS OF THE ACM*, 34:56–78, 1993.

[20] Christopher Dabrowski and Fern Hunt. Using markov chain analysis to study dynamic behaviour in large-scale grid systems. In *Proceedings of the Seventh Australasian Symposium on Grid Computing and e-Research - Volume 99*, AusGrid '09, pages 29–40, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[21] Carole Delporte-Gallet, Hugues Fauconnier, and Hung Tran-The. *Distributed Computing and Networking: 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings*, chapter Uniform Consensus with Homonyms and Omission Failures, pages 161–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[22] Qi Dong and Donggang Liu. Resilient cluster leader election for wireless sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1–9, June 2009.

[23] A. Dracinschi and S. Fdida. Congestion avoidance for unicast and multicast traffic. In *Universal Multiservice Networks, 2000. ECUMN 2000. 1st European Conference on*, pages 360–368, 2000.

[24] Z. M. Fadlullah, M. M. Fouda, N. Kato, A. Takeuchi, N. Iwasaki, and Y. Nozaki. Toward intelligent machine-to-machine communications in smart grid. *IEEE Communications Magazine*, 49(4):60–65, April 2011.

[25] N. Falliere, L. O. Murchu, and E. Chien. W32.Stuxnet Dossier. `http://goo.gl/dC8VT`, 2010. [Online; accessed December 2011].

[26] Michael Mackay Faycal Bouhafs and Madjid Merabti. Links to the future, jan 2012.

[27] Timothy French. *Bisimulation Quantifiers for Modal Logics*. PhD thesis, University of Western Australia, 2006.

[28] S. Galli, A. Scaglione, and Z. Wang. For the grid and through the grid: The role of power line communications in the smart grid. *Proceedings of the IEEE*, 99(6):998–1027, June 2011.

[29] H. Garcia-Molina. Elections in a distributed computing system. *Computers, IEEE Transactions on*, C-31(1):48 –59, January 1982.

[30] S. Ghosh. *Distributed Systems: An Algorithmic Approach*. Chapman & Hall, 2007.

[31] Aniruddha Gokhale, Mark P McDonald, Steven Drager, and William McKeever. A cyber physical systems perspective on the real-time and reliable dissemination of information in intelligent transportation systems. Technical report, DTIC Document, 2010.

[32] C. Gomez-Calzado, M. Larrea, I. Soraluze, A. Lafuente, and R. Cortinas. An evaluation of efficient leader election algorithms for crash-recovery systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, pages 180–188, 2013.

[33] F. Gmez-Cuba, F. J. Gonzlez-Castao, and C. P. Prez-Garrido. Practical smart grid traffic management in leased internet access networks. In *Energy Conference (ENERGYCON), 2014 IEEE International*, pages 852–858, May 2014.

[34] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, July 1990.

[35] N. Higgins, V. Vyatkin, N. K. C. Nair, and K. Schwarz. Distributed power system automation with iec 61850, iec 61499, and intelligent control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(1):81–92, Jan 2011.

[36] Gerry Howser and Bruce McMillin. Modeling and reasoning about the security of drive-by-wire automobile systems. *International Journal of Critical Infrastructure Protection*, pages 127 – 134, 2012.

[37] Gerry Howser and Bruce McMillin. A Multiple Security Domain Model of a Drive-by-Wire System. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 369–374. Computer Software and Applications Conference, 2013.

[38] S. Jackson and B. M. McMillin. The effects of network link unreliability for leader election algorithm in a smart grid system. In *Critical Information Infrastructures Security*, pages 59–70. Springer, Berlin, Heidelberg, 2013.

[39] Stephen Jackson and Bruce McMillin. Markov models of leader elections in a smart grid system (under review). *Journal of Parallel and Distributed Computing*, 2015.

[40] M. C. Janssen, P. A. Crossley, and L. Yang. Bringing iec 61850 and smart grid together. In *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*, pages 1–5, Dec 2011.

[41] Dong Jin, D. M. Nicol, and Guanhua Yan. An event buffer flooding attack in dnp3 controlled scada systems. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2614–2626, Dec 2011.

[42] Joost-Pieter Katoen. Model checking meets probability: a gentle introduction. In *Engineering dependable software systems*, volume 34 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 177–205. IOS Press, Amsterdam, 2013.

[43] K. V. Katsaros, W. K. Chai, N. Wang, G. Pavlou, H. Bontius, and M. Paolone. Information-centric networking for machine-to-machine data delivery: a case study in smart grid applications. *IEEE Network*, 28(3):58–64, May 2014.

[44] K. S. Kishor. Sharpe, March 2014. `http://sharpe.pratt.duke.edu/`.

[45] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462, May 2010.

[46] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[47] S. Liu, X. P. Liu, and A. E. Saddik. Denial-of-service (dos) attacks on load frequency control in smart grids. In *Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES*, pages 1–6, Feb 2013.

[48] R. E. Mackiewicz. Overview of iec 61850 and benefits. In *2006 IEEE Power Engineering Society General Meeting*, pages 8 pp.–, 2006.

[49] Mai Gehrke and Hideo Nagahashi and Yde Venema. A Sahlqvist theorem for distributive modal logic. *Annals of Pure and Applied Logic* , 131(13):65 – 102, 2005.

[50] S. Meiling, T. C. Schmidt, and T. Steinbach. On performance and robustness of internet-based smart grid communication: A case study for germany. In *2015*

IEEE International Conference on Smart Grid Communications (SmartGridComm), pages 295–300, Nov 2015.

[51] N. Mohammed, H. Otrok, Lingyu Wang, M. Debbabi, and P. Bhattacharya. Mechanism design-based secure leader election model for intrusion detection in MANET. *Dependable and Secure Computing, IEEE Transactions on*, 8(1):89–103, Jan 2011.

[52] L.E. Moser, P.M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39:54–63, 1996.

[53] NSF FREEDM Systems Center. FREEDM, The Future Renewable Electric Energy Delivery and Management Systems Center. `http://www.freedm.ncsu.edu/`.

[54] P. Olofsson. *Probability, Statistics, and Stochastic Processes, 2nd Edition*. John Wiley & Sons, 2012.

[55] N. Privault. *Understanding Markov Chains*. Springer Singapore, 2013.

[56] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ecn) to ip, 9 2001. RFC 3168.

[57] D.B. Rawat, C. Bajracharya, and Gongjun Yan. Towards intelligent transportation cyber-physical systems: Real-time computing and communications perspectives. In *SoutheastCon 2015*, pages 1–6, April 2015.

[58] Robbert Van Renesse, Takako M. Hickey, and Kenneth P. Birman. Design and performance of Horus: A lightweight group communications system. Technical report, Cornell, 1994.

[59] Thomas Roth and Bruce McMillin. Breaking Nondeducible Attacks on the Smart Grid. In *Seventh CRITIS Conference on Critical Information Infrastructures Security*. Seventh CRITIS Conference on Critical Information Infrastructures Security, 2012. (to appear).

[60] R.A. Sahner and K.S. Trivedi. Sharpe: a modeler's toolkit. In *Computer Performance and Dependability Symposium, 1996., Proceedings of IEEE International*, page 58, Sep 1996.

[61] K. Sampigethaya and R. Poovendran. Cyber-physical system framework for future aircraft and air traffic control. In *Aerospace Conference, 2012 IEEE*, pages 1–9, March 2012.

[62] Matthias Schmalz, Daniele Varacca, and Hagen Völzer. *CONCUR 2009 - Concurrency Theory: 20th International Conference, CONCUR 2009, Bologna,*

Italy, September 1-4, 2009. Proceedings, chapter Counterexamples in Probabilistic LTL Model Checking for Markov Chains, pages 587–602. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[63] M.M. Shirmohammadi, K. Faez, and M. Chhardoli. Lele: Leader election with load balancing energy in wireless sensor network. In *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, volume 2, pages 106–110, Jan 2009.

[64] C. Singh and A. Sprintson. Reliability assurance of cyber-physical power systems. In *Power and Energy Society General Meeting, 2010 IEEE*, pages 1 –6, July 2010.

[65] M.J. Stanovich, I. Leonard, K. Sanjeev, M. Steurer, T.P. Roth, S. Jackson, and M. Bruce. Development of a smart-grid cyber-physical systems testbed. In *Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES*, pages 1–6, Feb 2013.

[66] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 261–272 vol.1, April 2003.

[67] Wenye Wang and Zhuo Lu. Survey cyber security in the smart grid: Survey and challenges. *Comput. Netw.*, 57(5):1344–1371, April 2013.

[68] Y. Yan, Y. Qian, H. Sharif, and D. Tipper. A survey on smart grid communication infrastructures: Motivations, requirements and challenges. *Communications Surveys Tutorials, IEEE*, PP(99):1 –16, 2012.

[69] Yichi Zhang, Weiqing Sun, Lingfeng Wang, Hong Wang, R. C. Green, and M. Alam. A multi-level communication architecture of smart grid based on congestion aware wireless mesh network. In *North American Power Symposium (NAPS), 2011*, pages 1–6, Aug 2011.

[70] Ziang Zhang and Mo-Yuen Chow. Incremental cost consensus algorithm in a smart grid environment. In *Power and Energy Society General Meeting, 2011 IEEE*, pages 1 –6, July 2011.

VITA

Stephen Curtis Jackson was born in Kansas City, Missouri. He received his Bachelor's degrees in Computer Engineering and Computer Science from Missouri University of Science and Technology in December 2010. Afterward, he joined the Computer Science Ph.D. Program in January 2011 under the Information Assurance GAANN fellowship with Dr. Bruce McMillin as his research advisor. His research interests are in Markov models, cyber-physical systems, and distributed systems.