Markov Models of Leader Elections in a Smart Grid System

Stephen Jackson

May 15, 2014

1 Introduction

FREEDM (Future Renewable Electric Energy Delivery and Management) is a smart grid project focused on the future of the electrical grid. Major proposed features of the FREEDM network include the solid state transformer, distributed local energy storage, and distributed local energy generation [1]. This vein of research emphasizes decentralizing the power grid, making it more reliable by distributing energy production resources. Part of this design requires the system to operate in islanded mode, where portions of the distribution network are segmented from each other.

The effects of these partitions are still not well understood. This is particularly true in a distributed cyber-physical system, in which partitions may occur in both the cyber and physical domains. As a result,it is not well known how either a cyber or physical faults will effect the other portion of the system. However, previously conducted research [6][16] indicates that cyber faults can cause a physical system to apply unstable settings.

This work presents the initial steps necessary to better understanding and planning for these faults. A new 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.

This discussion presents an approach that utilizes Markov chain to model a system's grouping behavior. These chains produce expectations with regard to not only how long a system can be expected to stay in a particular state but also much time it will be able to spend coordinating and doing useful work over a period of time. Using these measures, the behavior of the control system for the physical devices can be adjusted to prevent faults.

The next section describes both the FREEDM smart-grid system and its Distributed Grid Intelligence (DGI). Section 3 includes an overview the group management algorithm, and the state of the art for distributed systems. Section 4 discusses the architecture of the DGI, including the architecture of the message passing interface and the protocols used in the experiments, and the experimental setup. Section 5 outlines previously gathered results collected by running the DGI in a controlled environment. Section 6 explains the assumptions made and the Markov Chain's construction. Section 7 focuses on the accuracy of these models for predicting grouping behavior. Section 8 offers summary and conclusion.

2 FREEDM DGI

This study models the group management module of the FREEDM 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.

The DGI software consists of a central component, known as the broker. This broker is responsible for presenting a communication interface. It also furnishes any common functionality the system's algorithms may need. These algorithms, grouped into modules, work in concert to move power from areas of excess supply to excess demand.

DGI utilizes several modules to manage a distributed smart-grid system. Group management, the focus of this work, 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 scheduler in which each module is given a predetermined window of execution which it may use to perform its duties. When a module's time period expires, the next module in the line is allowed to execute.

3 Group Management

The DGI uses the leader election algorithm, "Invitation Election Algorithm," written by Garcia-Molina [8]. Originally published in 1982, this 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 DGIs 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 [8], many of election algorithms have been created. Each algorithm is designed to be well-suited to the circumstances it will deployed in. Specialized algorithms exist for wireless sensor networks [20][7], detecting failures in certain circumstances [21][13], and of course, transient partitions. Work on leader elections has been incorporated into a variety of distributed frameworks: Isis [4], Horus [18], Totem [14], Transis [3], and Spread [2] all have methods for creating groups. Despite this wide array 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 his 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 nodes will enter a recovery state and operate alone until they can identify another coordinator. Therefore, a leader and each of the members maintain 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 [9]. 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 in order to merge groups. This serves as a ping / response query for detecting 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. Of course, simple modifications could allow the member to suspect other members.

In this work it is assumed that a leader does not span two partitioned networks; if a group was able to form, all members have some chance of communicating with one another.

This study uses a metric to assess the performance of the system under duress. A distributed can only perform meaningful work when the processes can work together to perform physical migration. This means that there are two networks that affect the system's ability to do work: the physical flow network and the cyber communication network.

4 Experimental Setup

4.1 Broker Architecture

The DGI software used in this designed around a broker architectural specification. Each core functionality of the system was implemented within a module that was provided access to core interfaces. These interfaces provided functionality such as scheduling requests, message passing, and a framework to manipulate physical devices.

The Broker provided a common message passing interface that all modules could access. This interface was then used to pass information between modules. For example, the list of peers in the group was sent as a message.

Several of the distributed algorithms used in the software required the use of ordered communication channels. DGI provided a reliable ordered communication protocol (The sequenced reliable connection or SRC). It also offered a "best effort" protocol (The sequenced unreliable connection or SUC). This protocol was also FIFO (first in, first out), but provided limited delivery guarantees.

Simple message delivery schemes were used in order to avoid complexities introduced by using TCP. Constructing a TCP connection to a process that either had failed or was unreachable required a considerable amount of time. We elected to use UDP packets which do not have those issues, since the protocol is connectionless. UDP also allowed for the development of multiple protocols with various properties to evaluate which properties are desirable. Lightweight protocols which were best effort oriented were implemented to deliver messages within the requirements. The protocols listed here continued operating despite omission failures. They follow the assumption that not every message is critical to the operation of the DGI and that the channel did not need to halt entirely to deliver one of the messages.

4.2 Sequenced Reliable Connection

The sequenced reliable connection was a modified send and wait protocol that had the ability to stop resending messages and move on to the next one in the queue if the message delivery time

exceeded some timeout. The design of this protocol met several criteria:

- Messages needed to be delivered in order. A number of distributed algorithms rely on the assumption that the underlying message channel is FIFO.
- Messages could become irrelevant. Some messages may only have a short period in which they are worth sending. Beyond that time period, they should be considered inconsequential and thus, skipped. Message expiration times were used to address this issue. After a certain amount of time had passed, the sender would longer attempt to write that message to the channel. Instead, he would proceed to the next unexpired message and attach a "kill" value. The kill value was the sequence number of the last message the sender knows the receiver accepted.
- Every effort needed to be made to deliver a message while it was still relevant.

One adjustable parameter, the resend time, controlled how often the system would attempt to deliver a message it had not received an acknowledgment for. A resend function was periodically called in an attempt to redeliver lost messages to the receiver.

4.3 Sequenced Unreliable Connection

The SUC protocol was a best effort protocol: it employed a sliding window to deliver messages as quickly as possible. A window size was chosen, the sender can have up to that many outstanding messages at any given time. The receiver would look for increasing sequence numbers, and disregard any message that was of a lower sequence number than was expected. The purpose of this protocol was to implement a bare minimum: messages were accepted in the order they are sent.

The SUC protocol's resend time could be adjusted. Additionally, the window size was configurable. However the window size was left unchanged for the tests presented in this work.

The SUC protocol was developed because early hypothesis omission failures supposed that a lighter weight protocol might be more advantageous. The lightweight protocol also presented a

more complex behavior to model using Markov chains, since the nature of the protocol allowed for a race condition where a packet is lost.

4.4 Experimental Setup

Network unreliability was simulated by dropping datagrams from specific sources on the receiver side. Each receiver was given an XML file describing the prescribed reliability of messages arriving from a specific source. The network settings were loaded at run time and could be polled if necessary for changes in the link reliability.

"Lost" messages are dropped on the receiver side. Code was inserted in the datagram processing code of the broker. The broker would not deliver the message to the modules if the message is selected to be dropped. On receipt of a message, the broker's examines the source and randomly selected based on the prescribed reliability to drop a message. A dropped message was not delivered to any of the modules and was not acknowledged by the receiver. This method emulated a lossy network link but not one with message delays.

5 Previous Results

Initial data was collected from a non-real time version of the DGI code. For each selected message arrival chance, as many as forty tests were run. The collected results from the tests are divided into several target scenarios as well as the protocol used.

The first minute of each test in the experimental test is 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 in group time. These graphs first appeared in [10].

5.1 Sequenced Reliable Connection

5.1.1 Two Node Case

The 100ms resend SRC test with two nodes can be considered a type of control in this study. These tests, pictured in Figure 1, highlight the performance of the SRC protocol. The maximum in group time of 9 minutes was achieved with only 15% of datagrams arriving at the receiver.

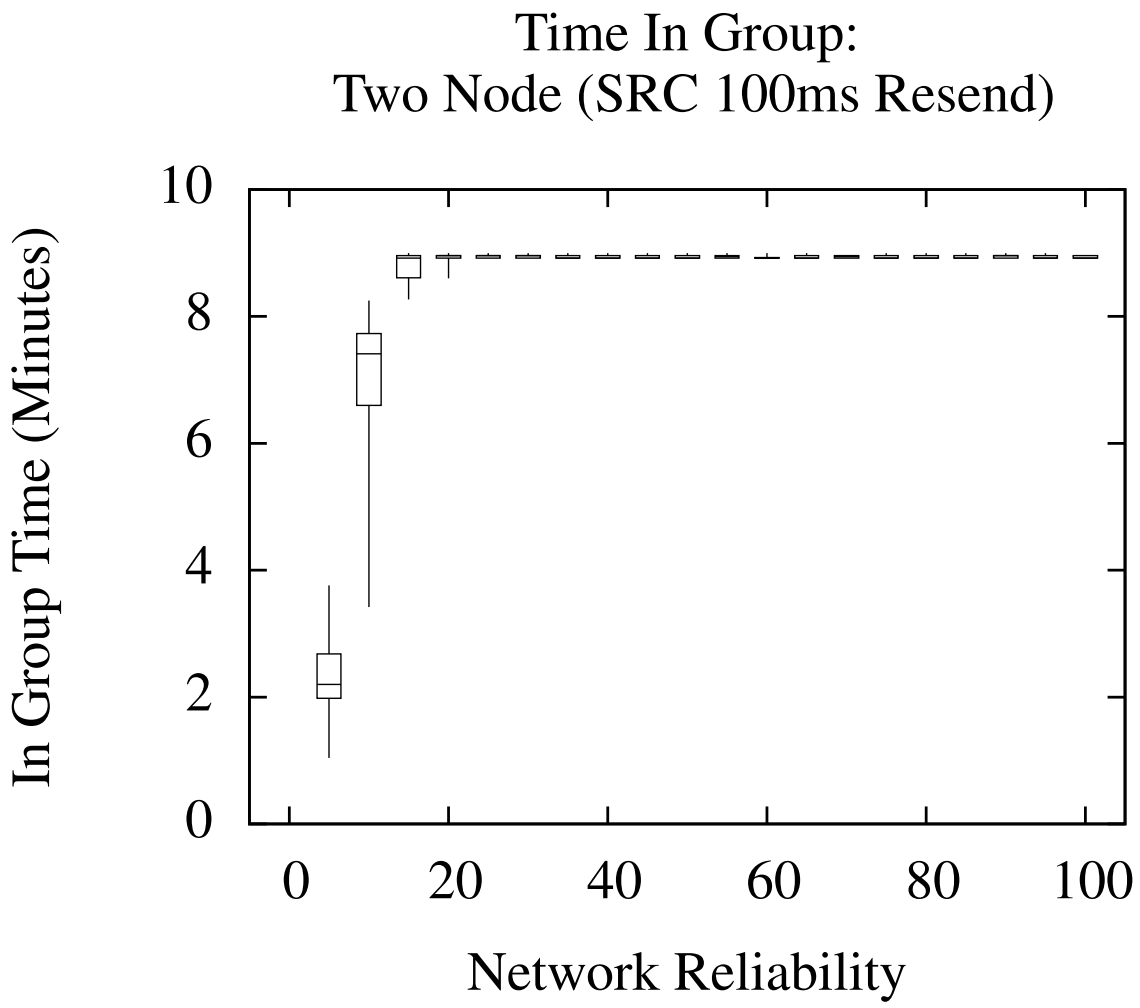

Figure 1: Time in-group over a 10 minute run for a two node-system with a 100ms resend time

Figure 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.

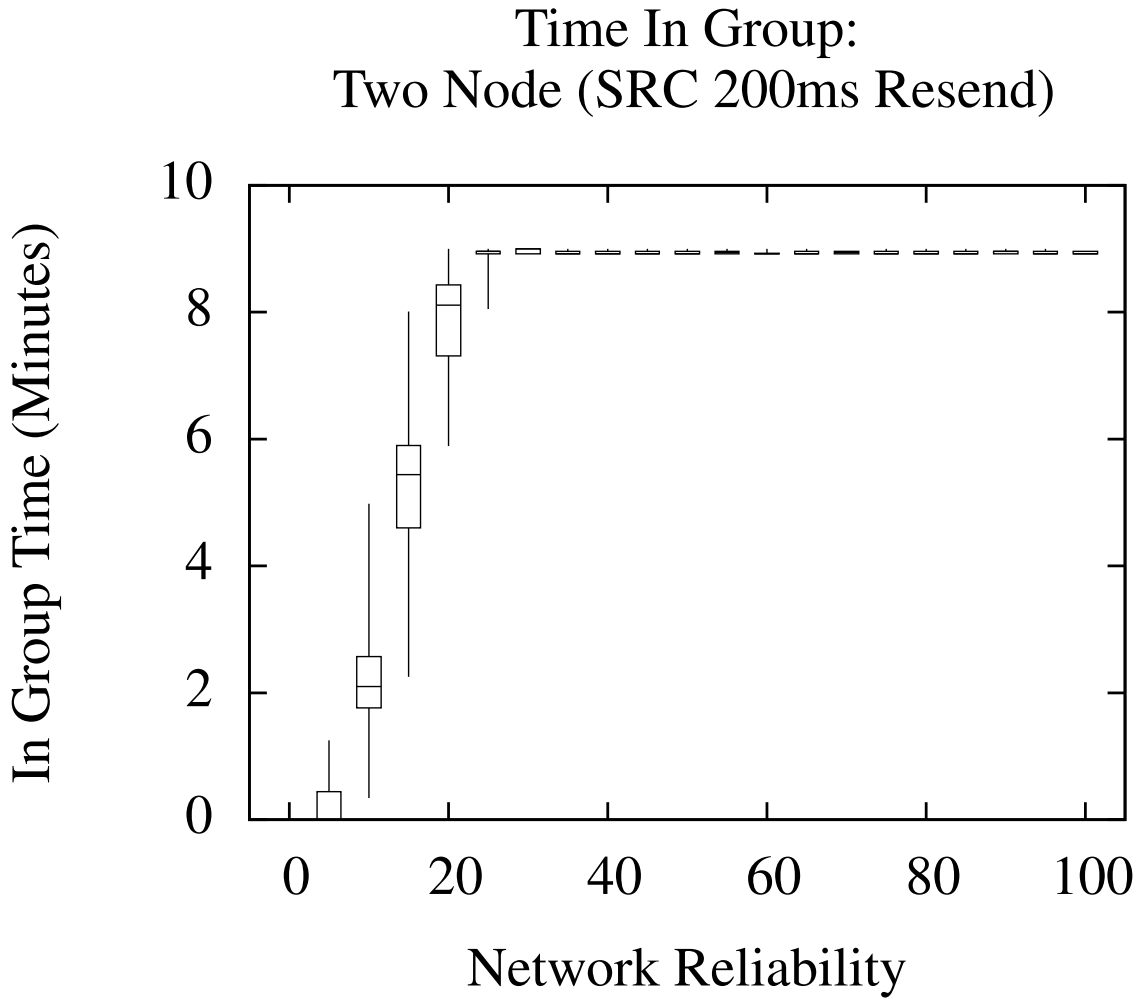

Figure 2: Time in-group over a 10 minute run for a two node-system with a 200ms resend time

5.1.2 Transient Partition Case

The transient partition case is 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 is unreachable, nodes will form a group with the other nodes on the same side of the partition. In this study, two processes were present on each side of the partition. In the experiment, the probability of a datagram crossing the partition was increased as the experiment continued. The 100ms case is shown in Figures 3 and 4.

While messages cannot cross the partition, the DGIs stay in a group with the nodes on the

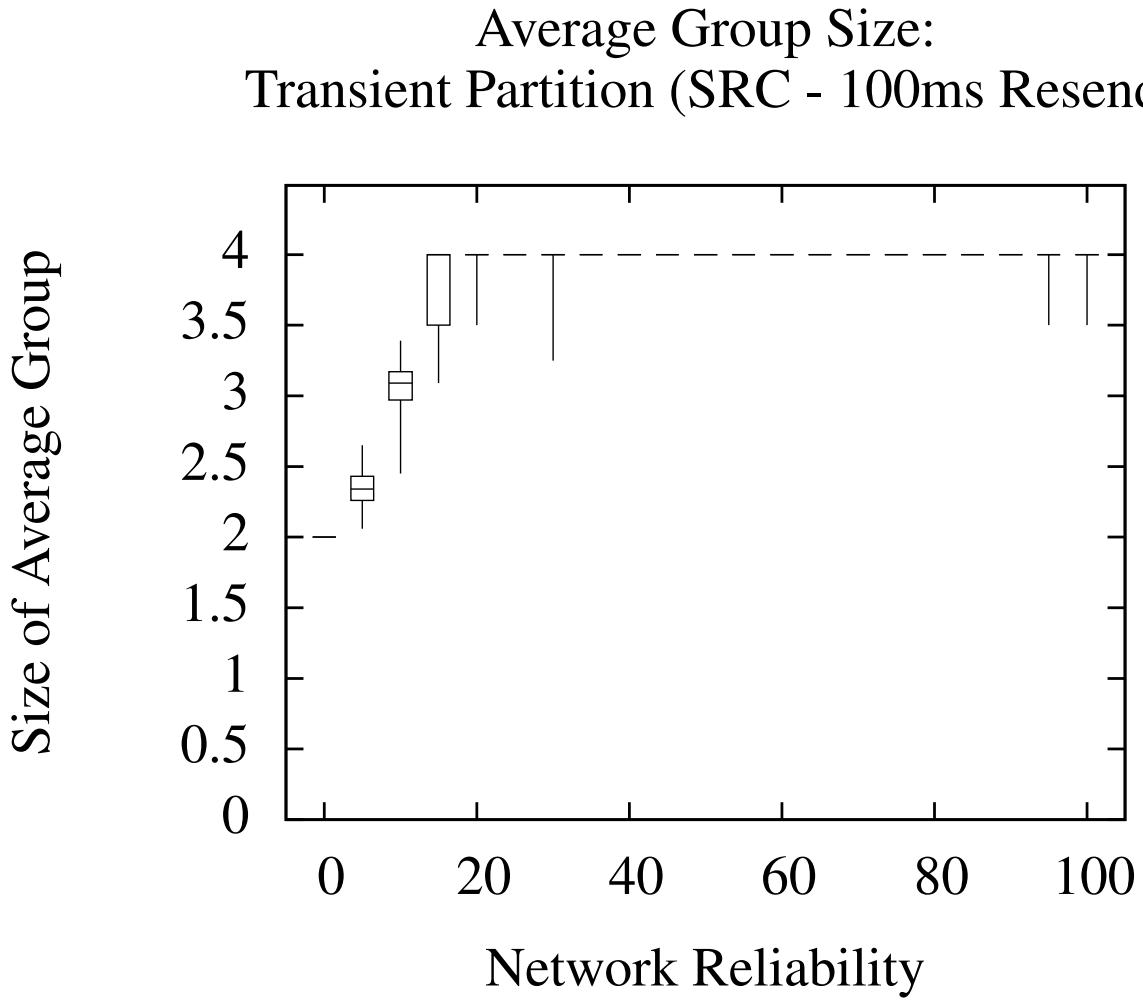

Figure 3: Average size of formed groups for a transient partition case with a 100ms resend time

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 reliability increases), DGI instances on either side attempted to complete elections with the nodes 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 in group time comes from elections.

The 200ms case (Illustrated in Figures 5 and 6) suggests a similar behavior to Figures 3 and 4,

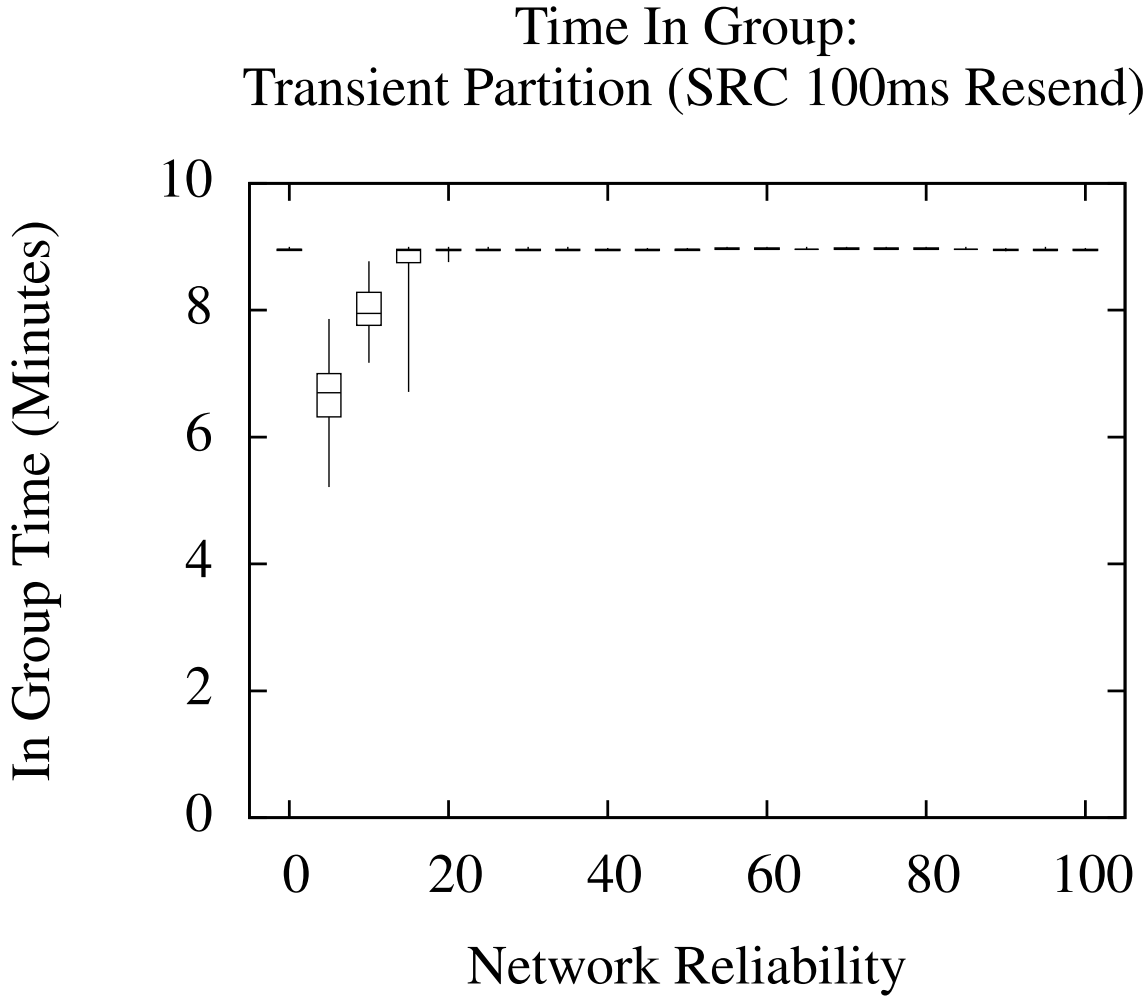

Figure 4: Time in-group over a 10 minute run for a transient partition case with a 100ms resend time

with a wider valley produced by the reduced number of datagrams. The mean group size dips below 2 in Figure 5, possibly because longer resend times allowed for a greater number race conditions between potential leaders. Discussion of these race conditions is shown and discussed during the SUC section since it was more prevalent in those experiments.

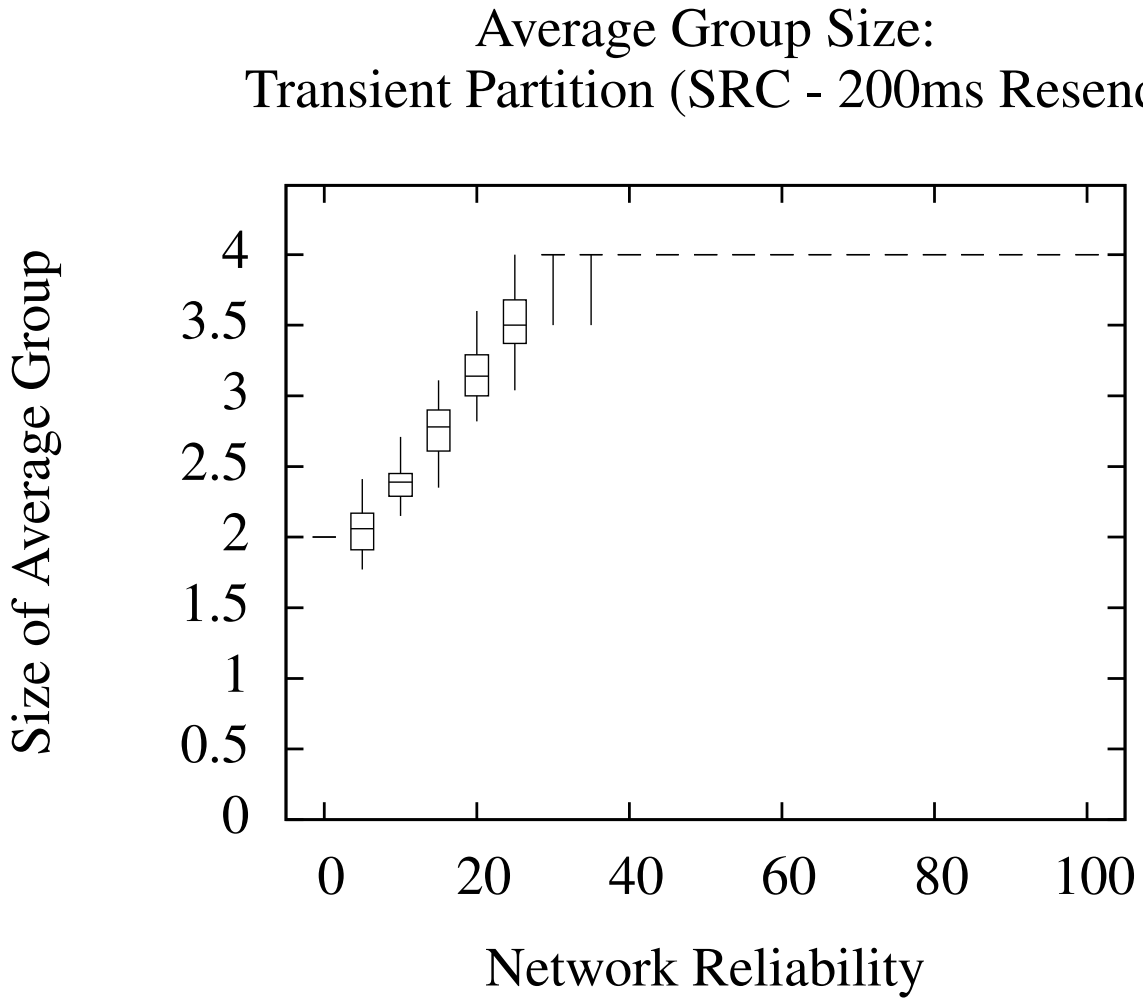

Figure 5: Average size of formed groups for a transient partition case with a 200ms resend time

5.2 Sequenced Unreliable Connection

5.2.1 Two Node Case

The SUC protocol's experimental tests revealed an immediate problem. There is a general increasing trend for the amount of time in-group shown in Figure 7. There is a high amount of variance, however, for any particular trial.

In the 200ms resend case (illustrated in Figure 8), a greater growth rate occurred in the in group time as the reliability increased. When an average was taken across all of the collected data points from the experiment, the average in group time is higher for the 200ms case than it

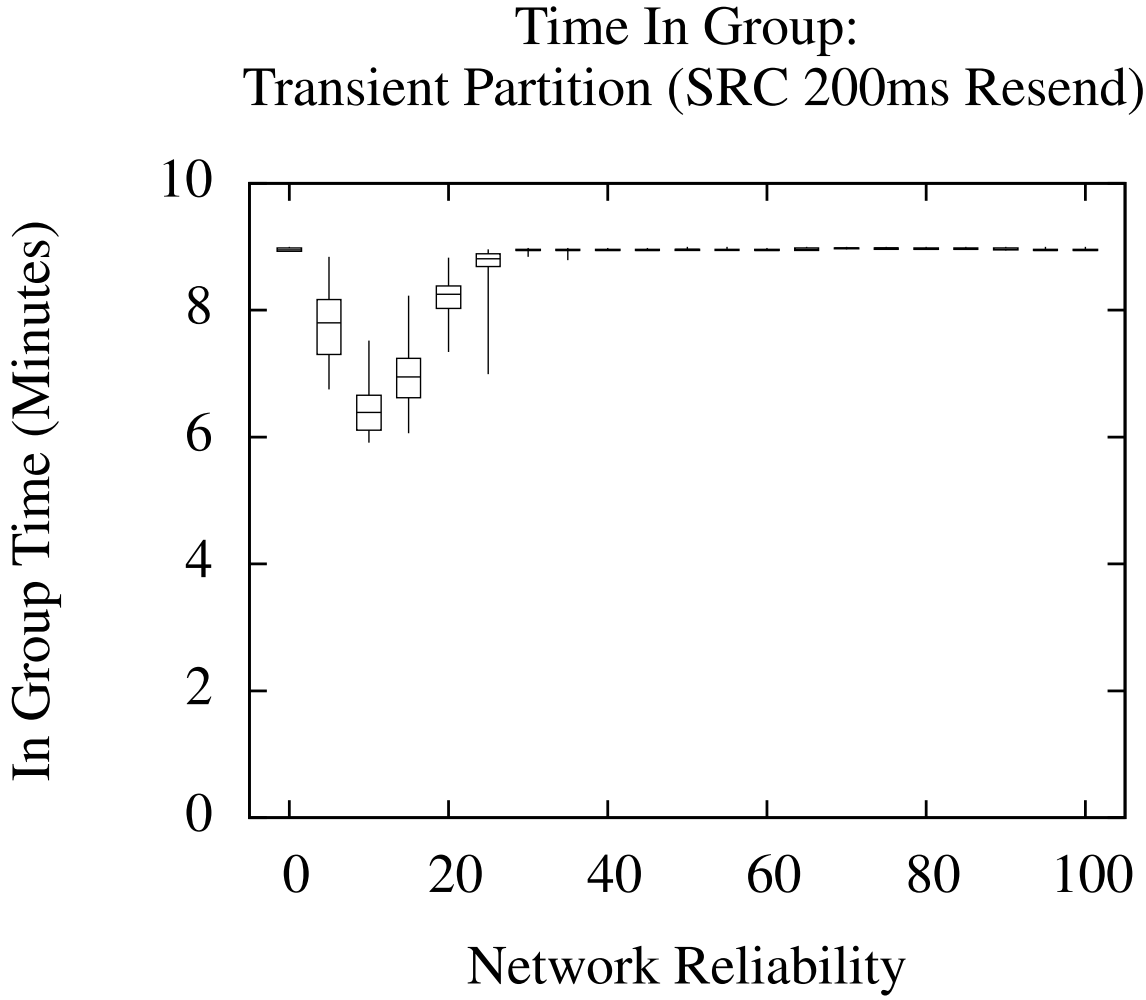

Figure 6: Time in-group over a 10 minute run for a transient partition case with a 200ms resend time

was for the 100ms case (6.86 vs 6.09). There large amount of variance in the collected in group time, however. As a result, it is not possible to state with confidence that the there is a significant difference between the two cases.

6 Formal Modeling

As a consequence of the high amount of variance in the collected data, it is difficult to make any predictions about other configurations from the collected data. The behavior of the DGIs

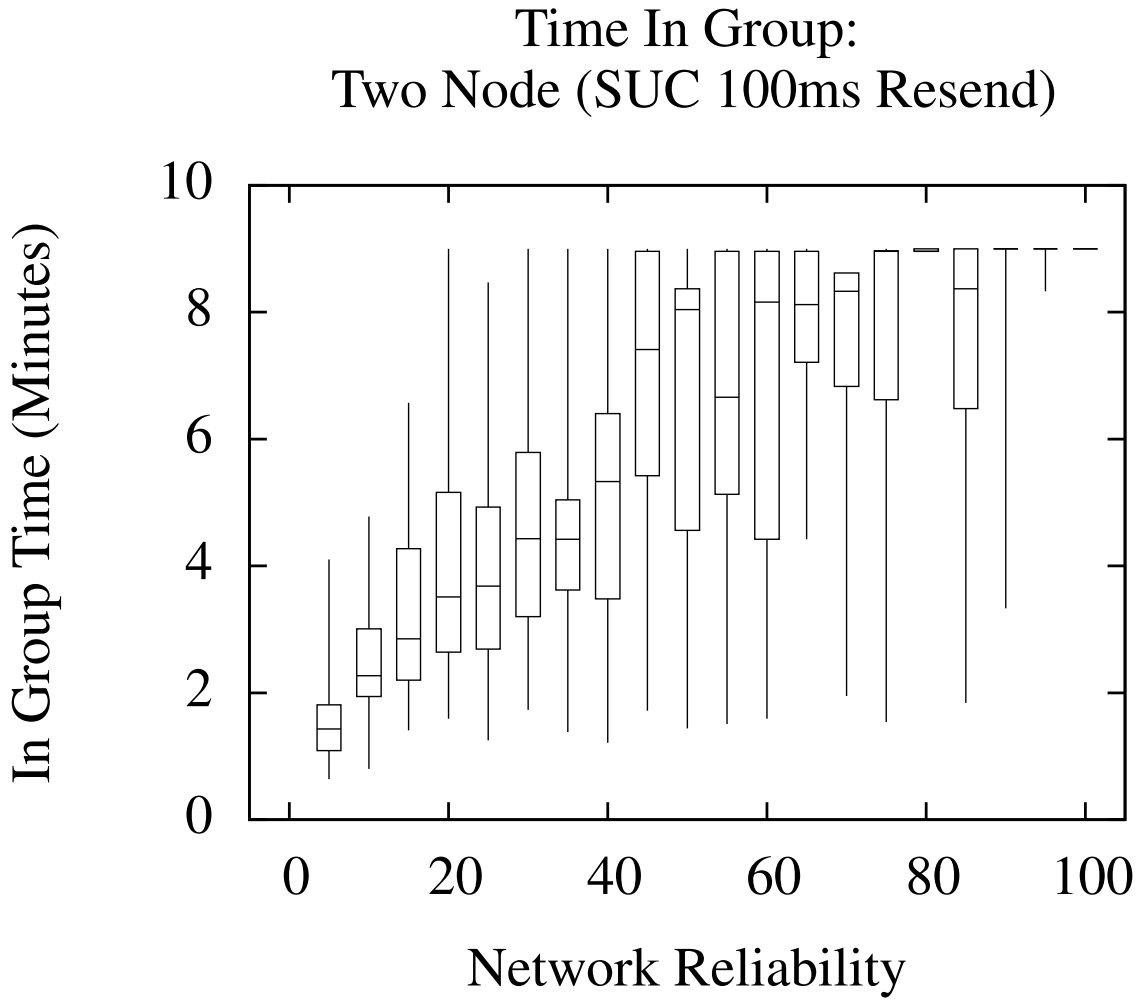

Figure 7: Time in-group over a 10 minute run for a two node system with a 100ms resend time

acting together was be modeled as a collection of states. Each state described configuration of the system, and that the transitions between those states model failure events or election events. These transitions are probabilistic, and as a result it is a natural extension to model the distributed system as a Markov Chain.

6.1 Assumptions

The inter-arrival time between events was assumed to be exponentially distributed. Furthermore, it was assumed that the system would be relatively well synchronized, with most elections occurring

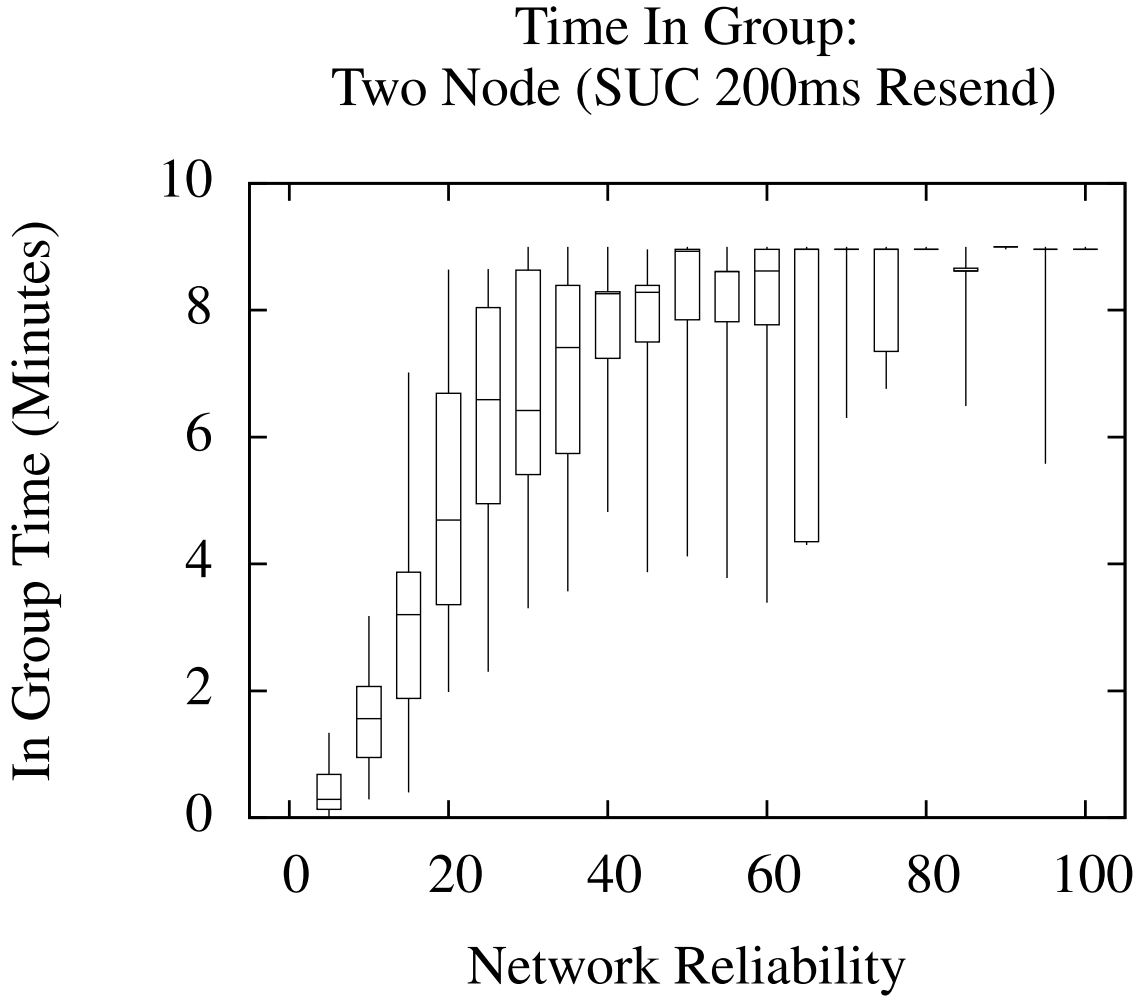

Figure 8: Time in-group over a 10 minute run for a two node system with a 200ms resend time

at the same time. This assumption was valid for the 2-node cases in the non-real-time code. An increased number of processes violated this assumption for the non-real-time code. With the use of the round-robin scheduler with synchronization it was possible enforce the synchronization assumption with the real-time code.

All participating peers were assumed to be on the same schedule; all peers began executing the model simultaneously. Synchronization was accomplished using Choi's work [5]. It was also assumed the clocks are synchronized. If the network has faulted, process clocks would not drift noticeably from their last synchronization. As an alternative, a production system would likely use

GPS time synchronization to obtain certain power system readings [12].

6.2 Constructing The Markov Chain

Consider a set of processes that are linked by some packet-based network protocol. Under ideal conditions, a packet sent by one process will always be delivered to its destination. Without a delivery protocol, as soon as packets are lost by the communication network, the message that it contained is lost forever. Therefore to compensate for packet loss, a large variety of delivery protocols have been developed. Each protocol has a different set of goals and objectives, dependent on the application. Two protocols, each with different delivery characteristics, were used in this study.

A single lost packet does not necessitate that the message it contained is forever lost. Different protocols allow for different levels of reliability, despite packet loss.

The leader election algorithm was centered around two critical events: checking and elections. The check system is used to detect both crash-stop failures and the availability of processes for election. Processes in the system occasionally exchanged messages to determine if the other processes had failed. Leaders exchanged messages to discover new leaders.

The DGI could perform work if in a group, and not in an election state. The group management module instructs other modules to stop during an election. The collected data in the previous sections was based on that assumption. The Markov chains that model those scenarios also use that assumption.

Events in the distributed system were assumed to be distributed exponentially. These were modeled in the chain by $\lambda(x)$ the parameter of the exponential distribution. [17][15] It is important to note that

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

$$\lambda(x) = \sum \lambda(x, y) = \sum \lambda(x) p(x, y) \tag{2}$$

where $\lambda(x)$ is the exponential parameter for the total time spent in a state x, $\lambda(x, y)$ is the expo-

nential parameter for a transition from state x to state y, and $p(x, y)$ is the normally distributed probability of transitioning from state x to state y.

6.2.1 Failure Detection

When a leader sent its check messages, the processes either responded in the positive (indicating that they are also leaders) or in the negative (indicating that they have already joined a group). This message was sent to all known nodes in the system. If a process replied that it was also a leader, the original sender entered an election mode and attempted to combine groups with the first process. Members that failed to respond were removed from the leader's group.

In contrast, the member only sent a check message to the leader of its current group. As with the leader's check message, the response could be either positive or negative. A "yes" response indicated that the leader is still available and considered the member a part of its group. A "no" response indicated that either the leader had failed and recovered, or it has suspected the member process of being unreachable. In either case, the member had been removed from the leader's group. The member would enter a recovery state and reset itself to an initial configuration where it was in a group by itself.

When a change in membership occurred, either due to recovery or a suspected failure, the leader pushed a list of members to the group. Members cannot suspect other members of failing. Only the leader can identify failed group members.

Figure 9 illustrates the model of the failure detection stage of the leader election algorithm. A set of processes begin in a normal state as part of a group. The leader sent a query to every member, and every member sent a query to the leader. If a response was not received in either direction, the process was considered to be unreachable. If the original message was sent by the leader, the leader removed that member and informed the remaining members. Otherwise, the member that sent the original message entered a recovery state.

The system remained in the original state as long as all nodes completed their queries and responses. Let T_R be the amount of time allowed for a response, T_C be the time between discovery

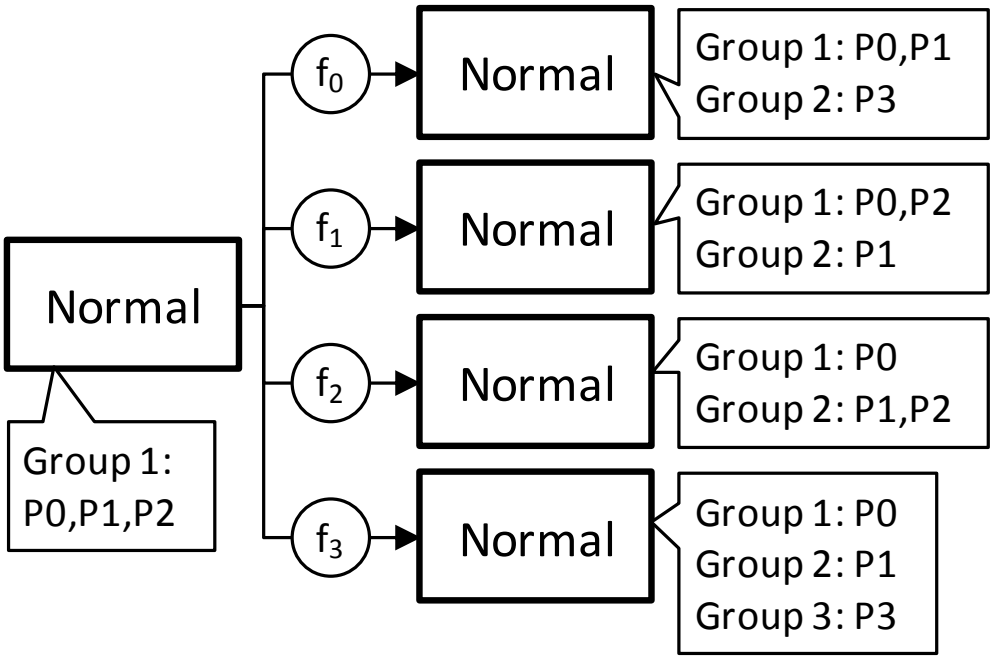

Figure 9: A diagram showing a partial Markov chain for failure detection

attempts, and p_F be the probability that at least one peer failed to complete the exchange. Based on this, the expected amount of time in the grouped state (T_G) was

$$\begin{cases} T_G = (T_R + T_C)/p_F & p_F > 0 \\ \infty & p_F = 0 \end{cases} \tag{3}$$

Let λ_G equal the exponential parameter of the exponential distribution for the expected time in some set of groups. The probabilities of each possible transition can then be related to the parameter for that configuration. Let p_i be the probability of transitioning to some set of groups i and let f_i be the exponential parameter for the transition to some other configuration:

$$\lambda_G = \sum f_i = \sum \delta p_i = \frac{1}{T_G} \tag{4}$$

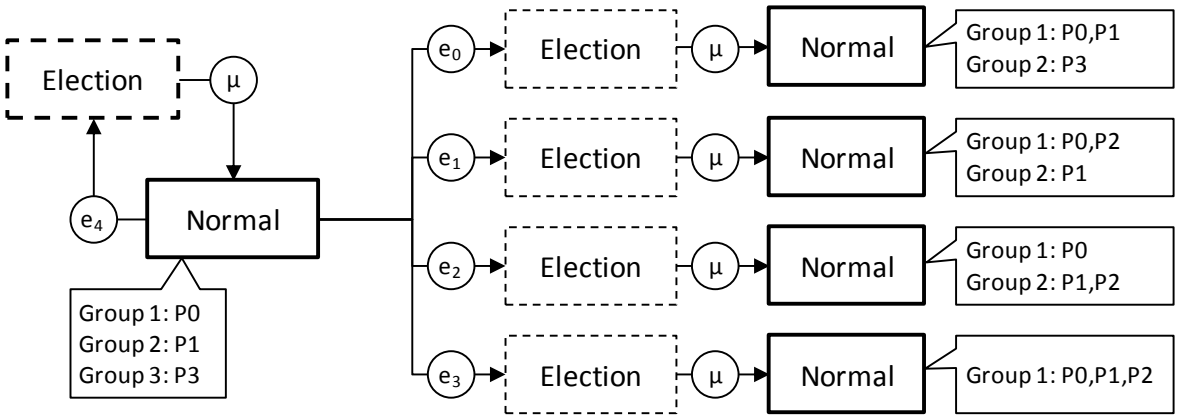

Figure 10: A diagram showing a partial Markov chain for an election

6.2.2 Leader Election

During elections, a highest priority leader (identified by its process ID) sent invites to the other leaders it had previously identified. If those leaders accepted the highest priority leader's invite, they replied with an accept message and forwarded the invite to their members. If the highest priority process failed to become the leader the next highest will sent invites after a specified interval had passed.

Therefore, the membership of the system was affected in two ways: 1) election events that changed the size of groups and 2) failure suspicions (via checks) which decreased the group's size. Note that elections could decrease the group's as well as increase them. If a round of forwarding invites fails, the group size could decrease.

When a process is initialized, it begins in the "solo" state. The solo state is a group in which it is the only member. The processes combine into groups as other processes are discovered by checks. Groups are not limited to increasing one at a time; they can increase by combined size of the groups of the leader processes.

A continuous time Markov model of a single election is presented in Figure 10. Here, a set of leaders begin in a normal state. After some time (T_D), an "are you coordinator" message discovers some other peer. T_D is a function of the number of discovery checks that do no discover any

leaders. This is a function of the link reliability. Let T_R be the amount of time allowed for a response, T_C be the time between discovery attempts, and p_D be the probability that the exchange discovers a leader.

$$\begin{cases} T_D = (T_R + T_C)/p_D & p_D > 0 \\ \infty & p_D = 0 \end{cases} \tag{5}$$

The parameters (e_x) in Figure 10 are a function of T_D and the probability an election results in configuration x (p_x).

$$e_x = \frac{p_x}{T_D} \tag{6}$$

Once a leader has been discovered, the system transitions into an election state, based on the potential outcome, in which the peers hold an election to determine a new configuration. An election can either succeed or fail, as illustrated in Figure 10. This results in a new system configuration, or each involved process returning to the single member group state.

The amount of time that an election takes is fixed before the algorithm is executed. Let T_E be the mean time it takes to complete any election. Therefore,

$$\mu = \frac{1}{T_E} \tag{7}$$

6.2.3 Combined Model

A combined model combines election and failure detection Markov chain components. Each state has a combination of election transitions and failure transitions. States where all reachable nodes are in the same group do not have election transitions. Likewise, states where there are no reachable leaders do not have election transitions. The combined model is predictive of the system's overall characteristics. The time spent in a particular configuration is a function of λ's for all events that can cause the system to transition away from a configuration.

Simulations of individual events were performed to construct the Markov chain. The circumstances for the events were assumed to be homogeneous; processes only differs by their process ID. Under this assumption, the simulation of events was broken down into a series of scenarios that represented of the system's events. Because each scenario is independent of other scenarios, each scenario ran independently. Additionally, because the circumstances were assumed to be homogeneous, scenarios that are similar such as ones where two processes swapped roles were simulated only once. The results were transformed from one scenario to another with a simple mapping. This mapping scheme and parallizability helped keep the state space explosion of the potential states under control.

7 Model Calibration

The presented methodology of constructing the model was initially calibrated against the original two-node case. This calibration used a non-real-time version of the DGI codebase. The resulting Markov chain was processed using SharpE [11][19] made by Dr. Kishor Trivedi's group at Duke University, 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 11 and 12.

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. This synchronization was the time difference between when each of them would search for leaders. The exchange of messages, particularly during an election, had a tendency to synchronize nodes during elections. Nodes 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.

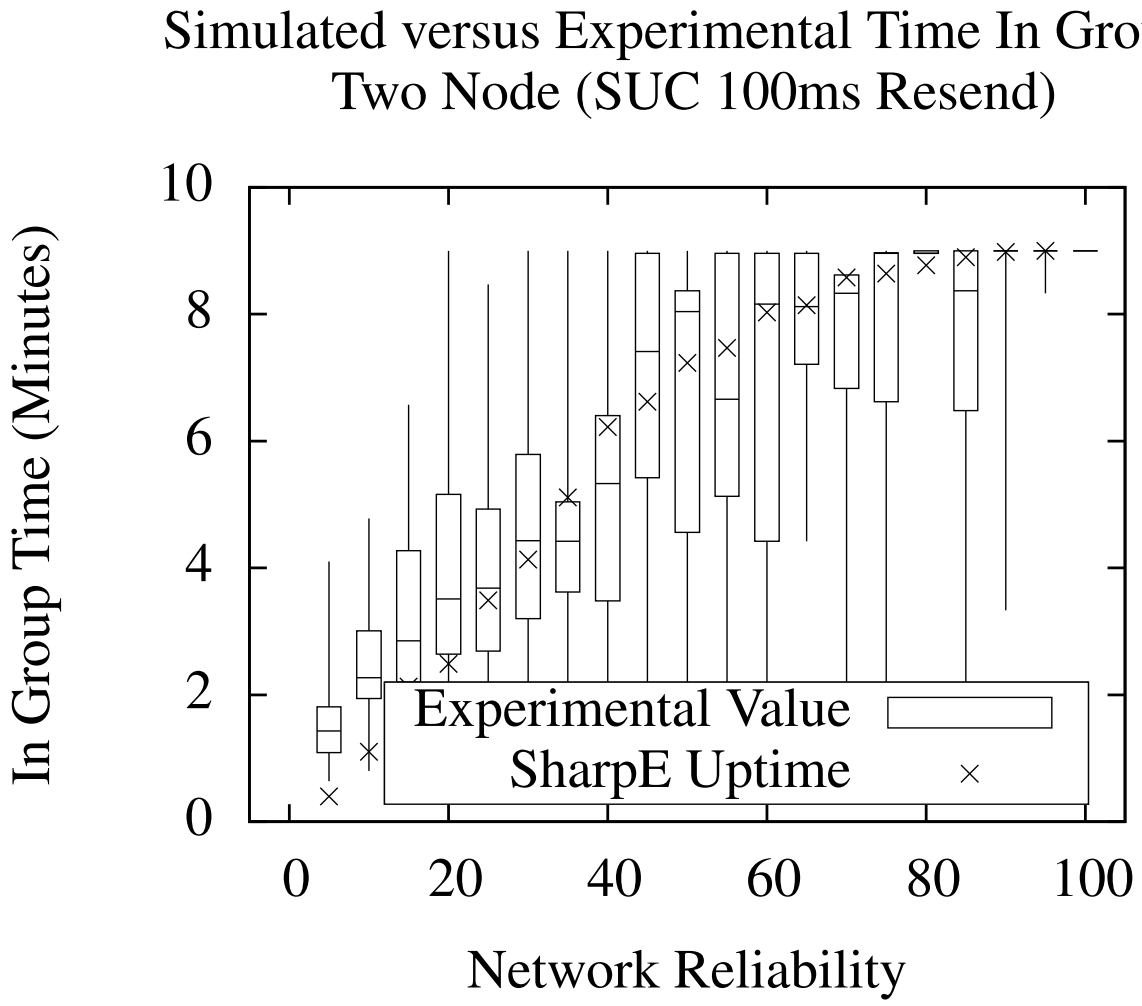

Figure 11: Comparison of in group time as collected from the experimental platform and the simulator (1 tick offset between processes).

Models fit to the non-real-time code in groups larger than two processes had a poor fit. This is presumed to be a combination of several factors. The major source of fault included the structure of the chain. Construction of the chain assumes that all processes enter the election state a roughly the same time. This was not typically true for more than two processes. Additionally, the simulator could only assume that the synchronization between processes was fixed. The coincidental synchronization that occurred in the two node case was suppressed by the increased number of peers. Furthermore, an issue with SharpE was discovered that prevented the particular structure of the chains produced from being handled correctly. The election states with only one outbound transi-

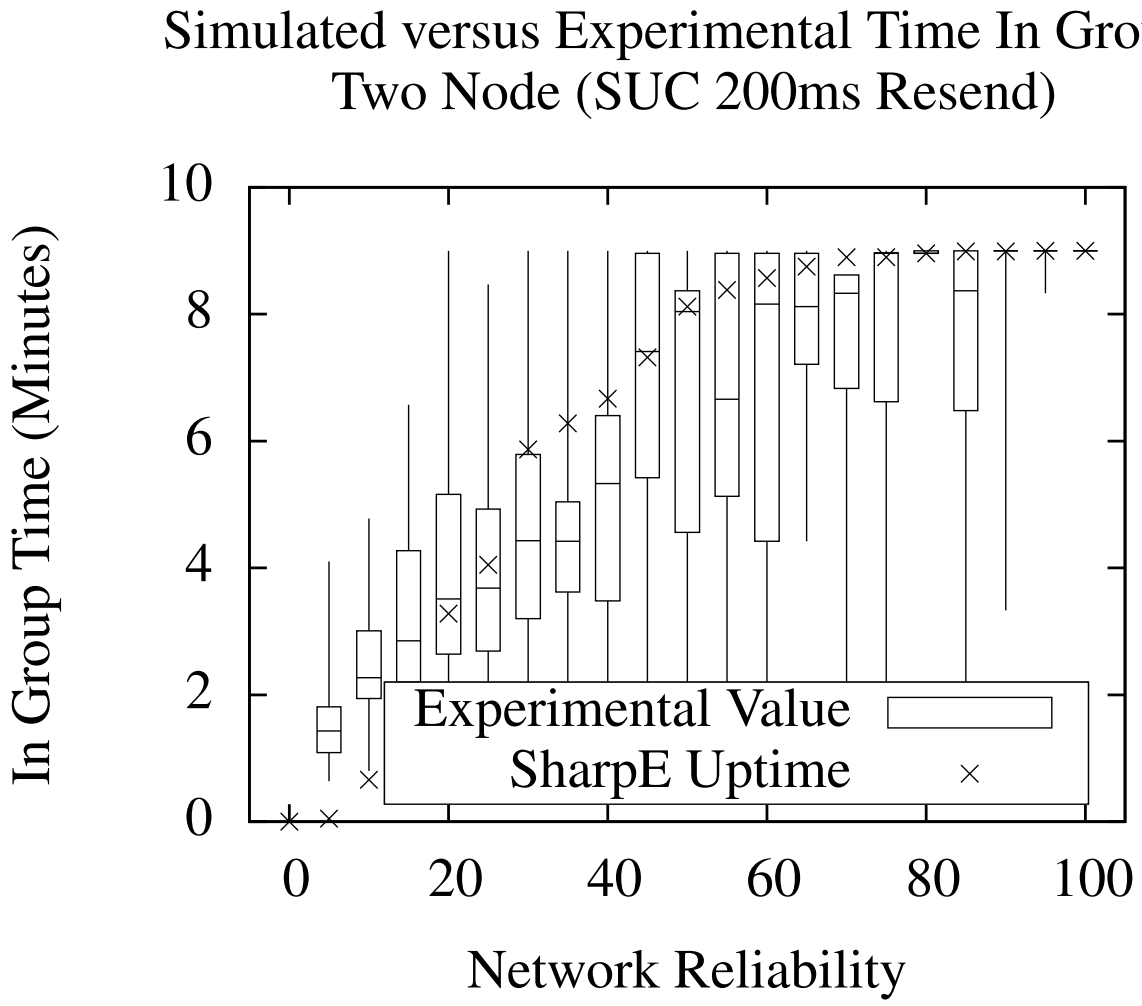

Figure 12: Comparison of in group time as collected from the experimental platform and the simulator (2 tick offset between processes).

tion uncovered a bug in the SharpE software. To circumvent that, issue, SharpE was replaced by a random-walker which generates exponentially distributed numbers and follows the paths of the chain. Time in group data for models which SharpE cannot process were collected across several hundred trials.

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 node, transient partition case. The collected data is overlaid with the results from the random walker in Figures 13 and 14.

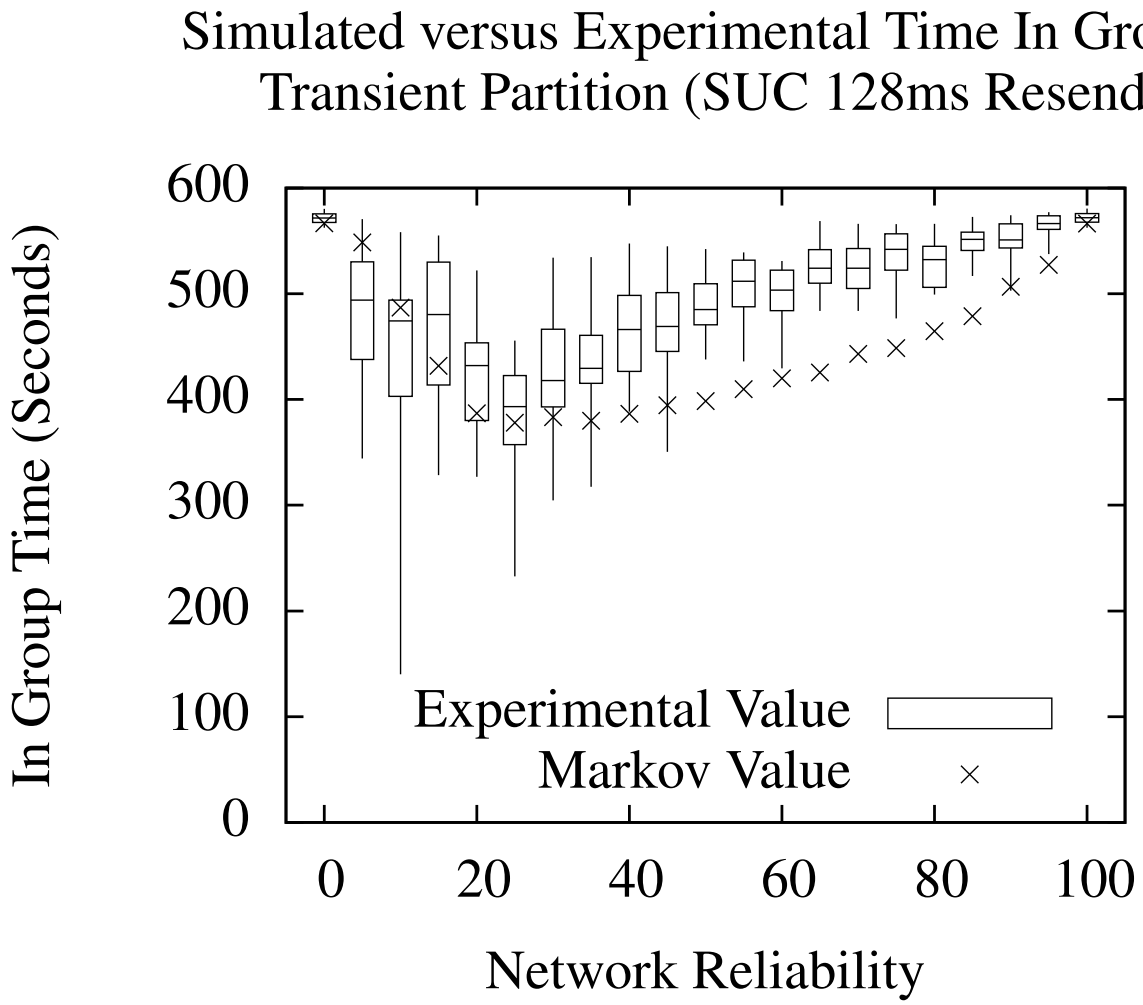

Figure 13: Comparison of in group time as collected from the experimental platform and the time in group from the equivalent Markov chain (128ms between resends).

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 1.

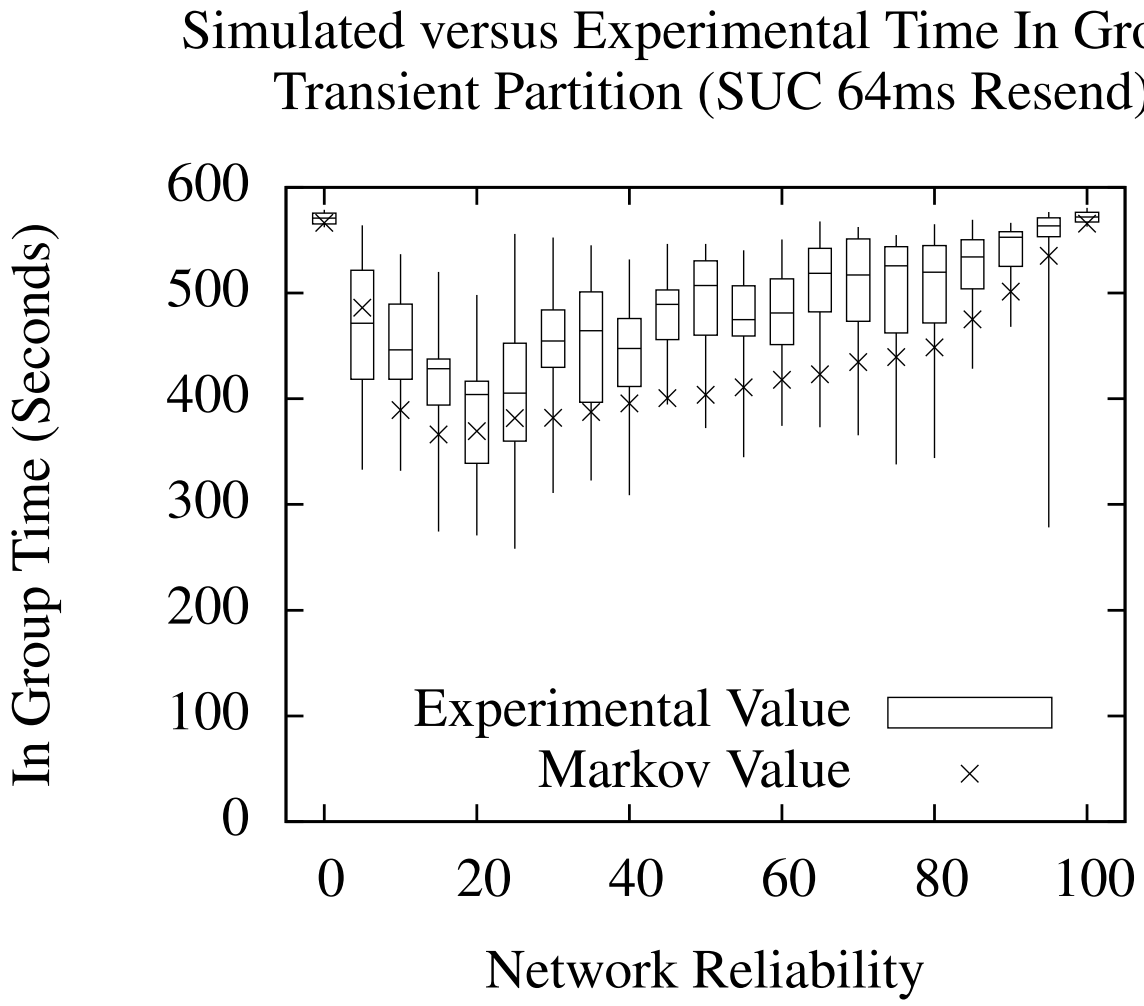

Figure 14: Comparison of in group time as collected from the experimental platform and the time in group from the equivalent Markov chain (64ms between resends).

8 Conclusion

This work presented a new approach for predicting the behavior of a real-time distributed system under omission failure conditions. Using a continuous time Markov chain, a variety of insights can be gathered about the system, including observations such as how long a particular configuration will be stable for and the behavior of the system in the long run. The Markov results will be used to make better real time schedules to react to the network faults we plan on introducing to our testbeds. For example, if migrations are failing and a sufficient number of migrations can cause

Table 1: Error and correlation of experimental data and Markov chain predictions

Resend	Correlation	Error
128	0.7656	11.61%
64	0.8604	11.70%

the physical system to fail, the scheduler may need to behave in a manner that limits the number of failed migrations. This work is a stepping stone towards designing a real-time schedule that manages the physical system correctly when there are cyber faults. Schedules and behavior can be designed around how the system behaves on its worst days. This work also allows these schedule designs and evaluations to be completed much more quickly than they could be by running the system for long periods of time. Therefore results from the test bed, combined with results the models yield schedules that improve the stability of the system during cyber faults.

References

[1] 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, Oct 2010.

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

[3] 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.

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

[5] Bong Jun Choi, Hao Liang, Xuemin Shen, and Weihua Zhuang. Dcs: Distributed asynchronous clock synchronization in delay tolerant networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(3):491–504, March 2012.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

[12] A. P S Meliopoulos, G.J. Cokkinides, O. Wasynczuk, E. Coyle, M. Bell, C. Hoffmann, C. Nita-Rotaru, T. Downar, L. Tsoukalas, and R. Gao. Pmu data characterization and application to stability monitoring. In *Power Engineering Society General Meeting, 2006. IEEE*, 2006.

[13] 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.

[14] 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.

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

[16] T. Paul, J.W. Kimball, M. Zawodniok, T.P. Roth, B. McMillin, and S. Chellappan. Unified invariants for cyber-physical switched system stability. *Smart Grid, IEEE Transactions on*, 5(1):112–120, Jan 2014.

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

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

[19] 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.

[20] 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.

[21] 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.