Markov Models of Distributed Systems for Smart Grid System

Stephen Jackson, *Member, IEEE* Bruce McMillin, *Senior Member, IEEE*

Abstract—Cyber-physical systems are an attractive option for future development of critical infrastructure systems. By supplementing the traditional physical network with cyber control, the performance and reliability of the system can be increased. In some of these networks, distributing the cyber control offers increased redundancy and availability during fault conditions. However, there are very few works which study the effects of cyber faults on a distributed cyber-physical system. These are of a particular interest in the Smart Grid environment where outages and failures are very costly. By examining the behavior of a distributed system under fault scenarios, the overall robustness of the system can be improved by planning characteristics and responses to faults that allow the system to continue operating in difficult circumstances.

Index Terms—Distributed Computing, Smart Grid, Leader Election, Group Management, Reliability, Robustness

1 Introduction

FREEDM (Future Renewable Electric Energy Delivery and Management) System 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, especially since in a distributed cyber-physical system partitions may occur in both the cyber and physical domains. As a result, it is not well known how cyber or physical fault will effect the other portion of the system. However, based on research such as [2], indicates that cyber faults, can cause a physical system apply unstable settings.

This work presents the initial steps to better understanding and planning for these faults. By taking a new approach to considering how a distributed system interacts during a fault condition, new techniques for managing a fault scenario in a cyber-physical systems will be created. To do this, we present an approach in modeling the behavior of a system using models and Markov chains. These chains produce expectations of how long a system can be expected to stay in a particular state, or how 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.

• *S. Jackson and B. McMillin are with the Department of Computer Science, Missouri University of Science & Technology, Rolla, MO, 65409. E-mail: scj7t4@mst.edu, ff@mst.edu*

2 FREEDM DGI

The FREEDM DGI (Distributed Grid Intelligence) is a smart grid operating system that organizes and coordinates power electronics and negotiates contracts to deliver power to devices and regions that cannot effectively facilitate their own need.

To accomplish this, the DGI software consists of a central component, the broker, which is responsible for presenting a communication interface and furnishing any common functionality needed by any algorithms used by the system. These algorithms are grouped into modules. These algorithms work in concert to move power from areas of excess supply to excess demand.

The DGI uses several modules to manage a distributed smart-grid system. Group management, the focus of this work, implements a leader election algorithm to discover which nodes are reachable in the cyber domain.

Other modules provide additional functionality such as collecting global snapshots, and a module which negotiates the migrations and gives commands to physical components.

The DGI is a real-time system: certain actions (and reactions) involving power system components need to be completed with a pre-specified time-frame to keep the system stable. The DGI uses a round robin scheduler: 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 in [3]. Orginally published in 1986, his algorithm provides a robust election procedure which allows for transient partitions. Transient partitions are formed when a faulty link between two or more clusters of DGIs causes the

groups to temporarily divide. These transient partitions merge when the link is 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 orginal publication, there has been a large body of work creating various election algorithms. Each algorithm is designed to be well suited to the circumstances it will deployed in: there are specialized algorithms for wireless sensor networks[4][5], detecting failures in certain circumstances[6][7], and of course, transient partitions. Work on leader elections has been incorporated into a variety of distributed frameworks: Isis[8], Horus[9], Totem[10], Transis[11], and Spread[12] all have methods for creating groups. Despite this wide body of work, the fundamentals of leader election are consistent accross all work: nodes arrive at a consensus of a single peer who coordinates the group, and nodes the 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 a up-to-date list of peers for the members of his group. Likewise, members of the group can detect the failure of 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 to join with. Therefore, a leader and each of the members maintains a set of processes which are currently reachable, which is a subset of all known processes in the system.

This Leader election can also be classified as a failure detector [13]. Failure detectors are algorithms which detect the failure of processes in a system. A failure detector algorithm maintains a list of processes that it suspects 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 members have separate roles to play in the failure detection process. The leader, will periodically search for other leaders to join groups with. This serves as a ping / response query for detecting failures in the system. It is also capable of detecting a change in state either by network issue or crash failure that causes the process being queried to no longer consider itself part of the leaders group. The member on the other hand, as the algorithm is written will only suspect the leader, and not the other processes. Of course, simple modifications could allow the member to suspect other members by use of a heart beat or query-reply system, it is not implemented in DGI code.

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

4 Experimental Setup

4.1 Broker Architecture

The DGI software is designed around a broker architectural specification. Each core functionality of the system is implemented within a module which is provided access to core interfaces which deliver functionality such as scheduling requests, message passing, and a framework to manipulate physical devices.

The Broker provides a common message passing interface that all modules are allowed to access. Information is passed between modules using this message passing interface. For example, the list of peers in the group is made available to other modules with a message.

Several of the distributed algorithms used in the software require the use of ordered communication channels. To achieve this, FREEDM provides a reliable ordered communication protocol (The sequenced reliable connection or SRC) to the modules, as well as a "best effort" protocol (The sequenced unreliable connection or SUC) which is also FIFO (first in, first out), but provides limited delivery guarantees.

We elected to design and implement our own simple message delivery schemes in order to avoid complexities introduced by using TCP in our system. During development, it was observed that constructing a TCP connection to a node that had failed or was unreachable took a considerable amount of time. We elected to use UDP packets which do not have those issues, since the protocol is connectionless. UDP also allows development of protocols with various properties to evaluate which properties are desirable. To accomplish this lightweight protocols which are best effort oriented were implemented to deliver messages as quickly as possible within the requirements.

The decision to go with a lighter weight protocol was also influenced by the FREEDM center targeting lower cost, less powerful ARM boards, with less available computing resources than a traditional server or desktop. Furthermore, the protocols listed here continue operating despite omission failures: they follow the assumption that not every message is critical to the operation of the DGI and that the channel does not need to halt entirely to deliver one of the messages.

4.2 Sequenced Reliable Connection

The sequenced reliable connection 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 exceeds some timeout. When designing this scheme we wanted to achieve several criteria:

- Messages must be delivered 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 certain amount of time has passed, the sender will no longer attempt to write that message to the channel. Instead, he 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 one adjustable parameter, the resend time, which controls how often the system would attempt to deliver a message it hadn't yet received an acknowledgment for.

Note that there is a resend function is periodically called to attempt to redeliver lost messages to the receiver.

4.3 Sequenced Unreliable Connection

The SUC protocol is simply a best effort protocol: it employs a sliding window to try to deliver messages as quickly as possible. A window size is decided, and then at any given time, the sender can have up to that many messages in the channel, awaiting acknowledgment. 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 also configurable, but was left unchanged for the tests presented in this work.

4.4 Experimental Setup

Network unreliability is 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.

On receipt of a message, the broker's communication layer examine the source and select randomly based on the reliability prescribed in the XML file whether or not to drop a message. A dropped message was not delivered to any of the sub-modules and was not acknowledged by the receiver. Using this method we were able to emulate a lossy network link but not one with message delays.

"Lost" messages are dropped on the reciever side. Code was inserted in the datagram processing code of the DGI. The DGI will not deliver the message to the modules if the message is selected to be dropped.

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 to remove any transients in the test. The result is that while the tests were run for ten minutes, the maximum result is 9 minutes of in-group time. These graphs first appeared in [14].

5.1 Sequenced Reliable Connection

5.1.1 Two Node Case

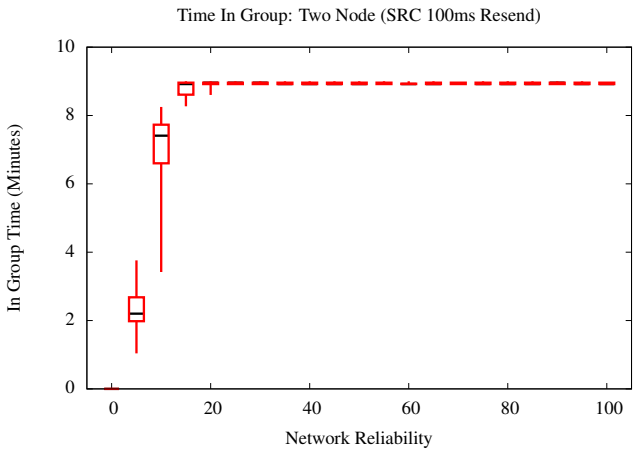

Fig. 1. Time in group over 10 minute run for two node system with 100ms resend time

The 100ms resend SRC test with two nodes can be considered a sort of a control. These tests, pictured in Figure 1. This test highlights the performance of the SRC protocol, achieving the maximum in-group time of 9 minutes with only 15% of datagrams arriving at the receiver.

Figures 2 demonstrates that as the rate at which lost datagrams are resent is decreased to resend every 200ms the time in group falls off. This behavior is expected, since each exchange has a time limit for each message to arrive and the number of attempts is reduced by increasing the resend time.

5.1.2 Transient Partition Case

The transient partition case shows a simple example where 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 our tests, there are two nodes on each side of the partition. In the experiment, the probability of a datagram crossing the partition is increased as the experiment continues. The 100ms case is shown in Figures 3 and 4.

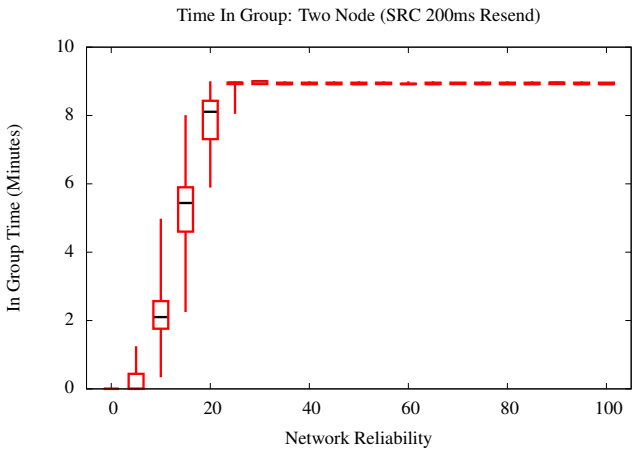

Fig. 2. Time in group over 10 minute run for two node system with 200ms resend time

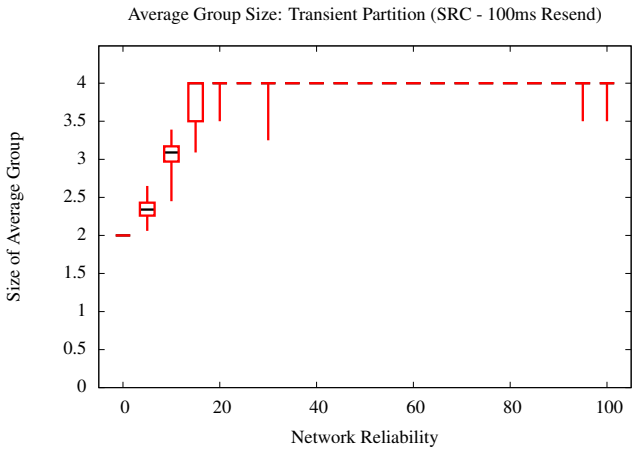

Fig. 3. Average size of formed groups for the transient partition case with 100ms resend time

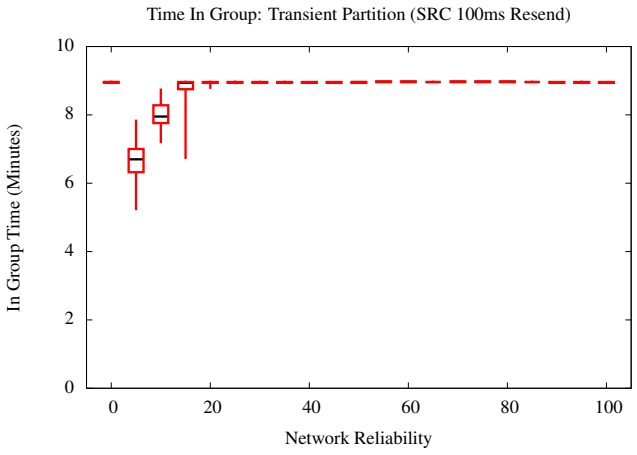

Fig. 4. Time in group over 10 minute run for the transient partition case with 100ms resend time

While messages cannot cross the partition, the DGIs stay in a group with the nodes on the same side of the partition leading to an in group time of 9 minutes, the maximum value. As packets begin to cross the partition (as the reliability increases), DGI instances on either side begin to attempt to complete elections with the nodes on the opposite partition and the time in group begins to fall. However during this time, the mean group size continues to increase, meaning while the elections are decreasing the amount of time that the module spends in state where it can actively do work, it typically does not fall into a state where it is in a group by itself, which means that most of the lost in-group time comes from elections.

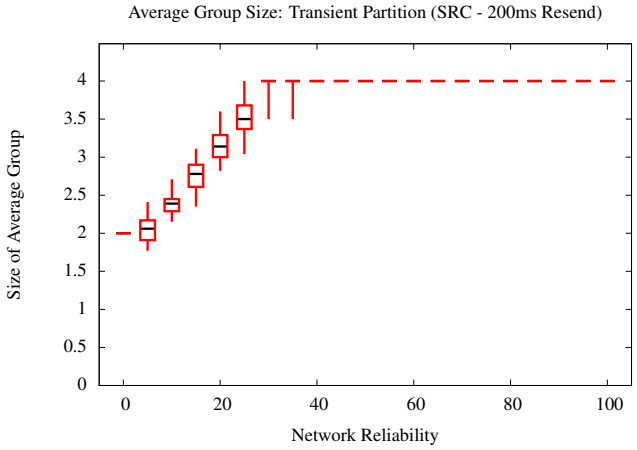

Fig. 5. Average size of formed groups for the transient partition case with 200ms resend time

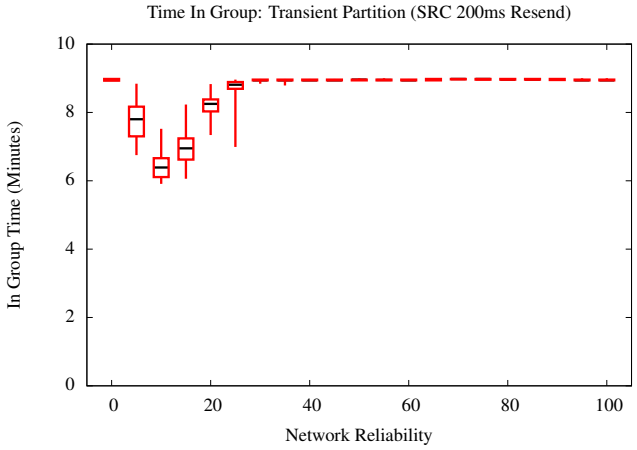

Fig. 6. Time in group over 10 minute run for the transient partition case with 200ms resend time

The 200ms case, shown in Figures 5 and 6 displays similar behavior, with a wider valley due to the limited number of datagrams. It is also worth noting the that the mean group size dips below 2 in the figure, possibly because the longer resend times allow for more race conditions between potential leaders. Discussion of these race conditions is shown in discussed during the SUC charts since it is more prevalent in those experiments.

5.2 Sequenced Unreliable Connection

5.2.1 Two Node Case

The SUC protocol's experimental tests show an immediate problem: although there is a general trend of growth in the amount of time in group and group size charts, shown in Figure 7 there is a high amount of variance for any particular trial.

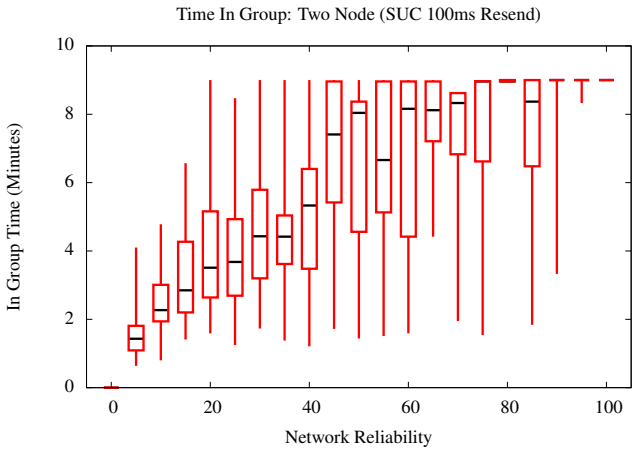

Fig. 7. Time in group over 10 minute run for two node system with 100ms resend time

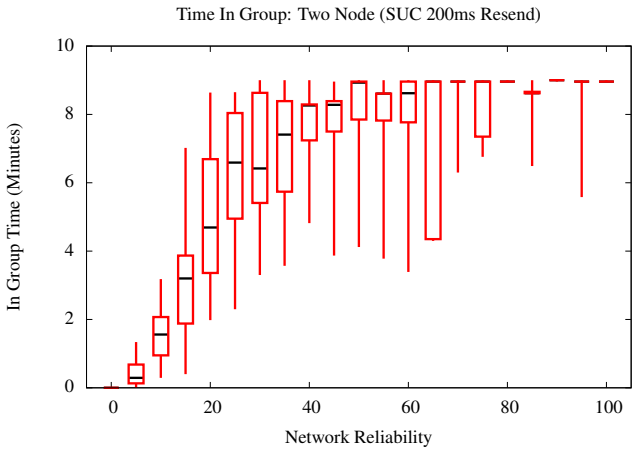

Fig. 8. Time in group over 10 minute run for two node system with 200ms resend time

In the 200ms resend case, show in Figure 8, it can be observed that there is a more growth rate in the in-group time as a the reliability increases. In fact, averaging across all the collected data points from the experiment, the average in-group time is higher for the 200ms case than it is for the 100ms case (6.86 vs 6.09). However, due to the large amount of variance in the collected in-group time, it is not possible to state with confidence that the there is a significant difference between the two cases.

6 Formal Modeling

Due to the high amount of variance in the collected data, and the resulting difficulty making any sort of prediction about other systems from the data, a more formal approach was tried.

Since the system, taken as a whole can be reasonably modeled as a collection of states each describing the state or configuration of the system, and that the transitions between those state (failure events or election events) are probabilistic, rather than deterministic, it is a natural extension to model the distributed system as a Markov Chain.

6.1 Assumptions

In order to model the system, we assumed that the time between events was exponentially distributed. Furthermore, we assumed that the system would be fairly well synchronized, with most elections occurring at the same time. This assumption was valid for 2-Node cases of our non-real-time code, but was a major issue as the number of nodes began to increase. However, with of the use of the round-robin scheduler with synchronization to enforce our real-time requirements, assuming the synchronization of processes is not a major leap.

All participating peers are assumed to be on the same schedule: all peers begin execution of a model simultaneously. This is accomplished using [15]. This work assumes that the clocks are synchronized: if the network has faulted, they have not drifted noticeably from their last synchronization. Additionally, a production system would likely use GPS time synchronization in order to take certain power system readings [16].

6.2 Constructing The Markov Chain

Consider a set of processes, which are linked by some packet based network protocol. In our experiments we provide two protocols, each with different delivery characteristics. 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 the network losing packets, a large variety of delivery protocols have been adapted. Each protocol has a different set of goals and objectives, depending on the application.

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

The leader election algorithm is centered around two critical events: checking, and elections. The check system is used to detect both failures and the availability of nodes for election. Processes in the system occasionally exchange messages to determine if the other processes have crashed, and to discover new leaders.

The DGI can perform work assuming that it is in a group, and not in an election state (since the Group

Management module instructs other modules to stop during an election). The collected data in the previous sections is based on that assumption, and the Markov Chains which models those scenarios needed to as well.

Processes in the DGI are either members or leaders. Leaders are processes which have won elections among its members.

As stated previously, it was assumed that the events in the distributed system were distributed exponentially. This was partially in order to facilitate the use of the program SharpE [17][18] made by Dr. Kishor Trivedi's group at Duke University, and partially for the ease of the mathematics involved in the continuous time Markov chain. Events are modeled in the chain using $\lambda(x)$ which is the parameter of the exponential distribution. It is important to note that:

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

and

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

Where $\lambda(x)$ is the exponential parameter for the total time spend in a state x, $\lambda(x, y)$ is the exponential parameter for a transition from state x to state y, and $p(x, y)$ is the normally distributed probability that a state transitions from state x to state y.

6.2.1 Failure Detection

When a leader sends its check messages, the nodes that receive it either respond in the positive, indicating that they are also leaders, or in the negative indicating that they have already joined a group. This message is sent to all known nodes in the system. If a process replies that it is also a leader, the original sender will enter and election mode and attempt to combine groups with the first process. Nodes that fail to respond are removed from the leaders group, if they were members.

The member on the other hand will only direct its check message to the leader of its current group. As with the leader's check message, the response can either be positive or negative. A yes response indicates that the leader is still available and considers the member a part of its group. A no response indicates that either the leader has failed and recovered, or it has suspected the member process of being unreachable (either due to crash or network issue) and has removed them from the group. In this event the member will enter a recovery state and reset itself to an initial configuration where it is in a group by itself.

On any membership change, either due to recovery, or a suspected failure, the list of members for a group is pushed to every member of that group by the leader. Members cannot suspect other processes of being crashed, only the leader can identify failed group members.

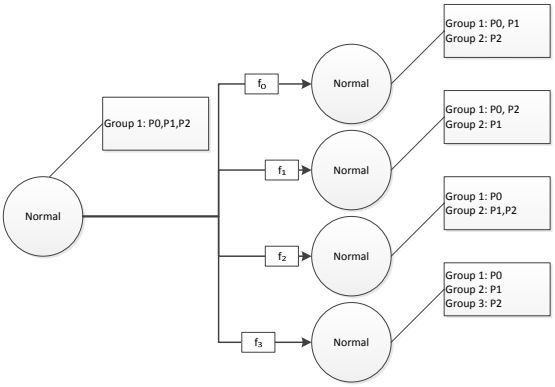

Fig. 9. A diagram showing a partial Markov chain for failure detection

A model of a failure detection stage of the leader election algorithm is presented in Figure 9. A set of nodes begin in a normal state as part of a group. The leader sends a query to every member, and every member sends a query to the leader. If a response is not received in either direction, the process is considered to be unreachable and is either ejected from the group by the leader (if the query originated from the leader) or the member leaves the group and becomes a coordinator themselves.

The system will stay in the original state as long as all nodes complete their queries and responses. Let T_R be the amount of time allowed for a response, T_C be the time between discovery attempts, and p_F is the probability that at least one peer fails to complete the exchange. Based on this, the expected amount of time in the grouped state (T_G) is:

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

Let δ equal exponential parameter of the exponential distribution for the base state. Then we can relate the probabilities of each possible transition to the parameter for the base state. Let p_i be the probability of transitioning to configuration i after leaving the base state and let f_i be the exponential parameter for the transition to an individual configuration:

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

6.2.2 Leader Election

During elections, a highest priority leader (identified by its process id) will send invites to the other leaders it has identified. If those leaders accept the highest priority leader's invites, they will reply with an accept message and forward the invite to their members, if their are any. If the highest priority process fails to become the leader the next highest will send invites after a specified interval has passed.

Therefore, the membership of the system can be affected in two ways: election events which change the size of groups and failure suspicion (via checks) which decreases the size of groups. Note that elections can decrease the size of groups as well as increase them: If a round of forwarding invites fails by the new leader to his original group, the group size could decrease.

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

We define a metric to assess the performance of the system under duress, we first consider that the 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 and the cyber.

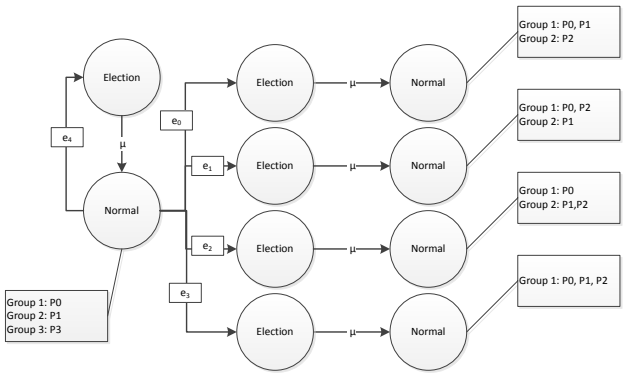

Fig. 10. A diagram showing a partial Markov chain for an election

A continuous time Markov model of a single election is presented in Figure 10. 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 which discover no leaders (which in turn 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 is the probability that the exchange discovers a leader.

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

Then, the parameters e_x in Figure 10 are a function of T_D and p_x, the probability an election results in configuration 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, where the peers hold an election to determine a new configuration. As shown in Figure 10, an election can either succeed or fail, resulting in a new system configuration, or each involved process to return 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. Except for the states where all reachable nodes are in the game group and the states where there are no reachable leaders each state has a combination of election transitions and failure transitions. The combined model is predictive of the overall characteristics of the system. The time spent in a particular configuration is a function of the λ's of all the events that can cause the system to transition away from a configuration.

To construct the Markov chain, simulations of individual events are performed. The circumstances for the events are assumed to be homogeneous: processes only differ by their process id. Using this assumption, the simulation of events can be broken down into a series of scenarios that are representative of the events in the system. Since each scenario is independent of other scenarios, each scenario can be run independently. Additionally, since the circumstances are assumed to be homogeneous, scenarios that are similar, such as ones where two processes swap roles can be simulated only once, and the results can be transformed from one scenario to another with a simple mapping. This mapping scheme and parallizability helps 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, using a non-real-time version of the DGI codebase. The resulting Markov chain was processed using SharpE which measured the reward collected in 600 second, minus the reward that was collected in the first 60 seconds (to emulate that the first 60 seconds 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 (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

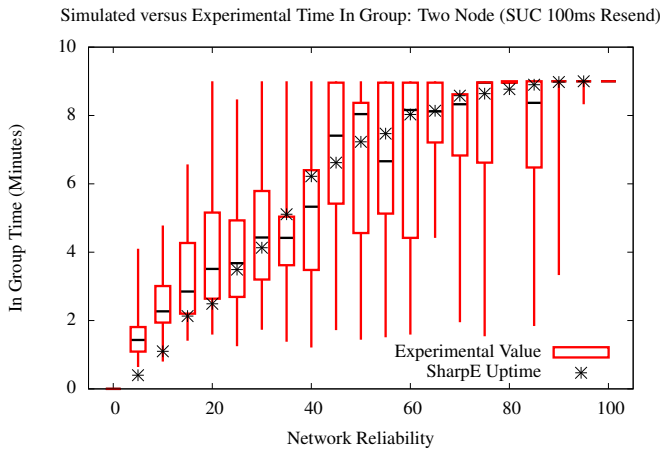

Fig. 11. Comparison of in-group time as collected from the experimental platform and the simulator (1 tick offset between processes).

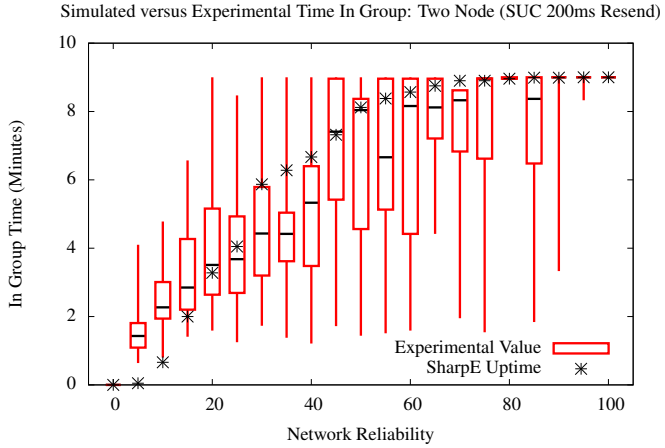

Fig. 12. Comparison of in-group time as collected from the experimental platform and the simulator (2 tick offset between processes).

elections, and so the nodes could synchronize even if they did not initially begin in a synchronized state. As a result, the simulation results aligned best for the 100ms resend case with 1 ticks (Approximately 100ms difference in synchronization between processes) and 2 ticks (Approximately 400ms) in the 200ms resend case.

Models fit to the non-real-time code in groups larger than 2 processes did not fit well. This is presumed to be a combination of several factors. The suspected major source of fault included the structure of the chain, which naturally assumes that all processes enter the election state a roughly the same time, which is not typically true for any number of processes greater than 2. Additionally, the simulator could only assume that the synchronization between processes was mostly fixed, which was not the case in the larger configurations, since 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. To circumvent that, issue, SharpE was replaced by a random-walker which generates exponentially distributed numbers and follows the paths of the chain, across several hundred trials, in order to collect time in group data for models which SharpE cannot process.

The structure of the Markov Chain, which assumed that processes enter the election state mostly simultaneously was an appropriate assumption for the real-time system, since the round-robin scheduler synchronizes when processes run their group management modules. The simulator was set to assume that the synchronization between processes was very tight, and 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.

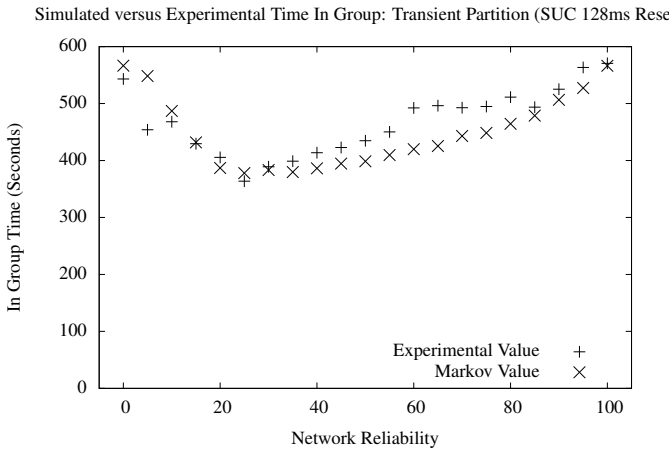

Fig. 13. Comparison of in-group time as collected from the experimental platform and the time in group from the equivalent Markov chain (128ms between resends).

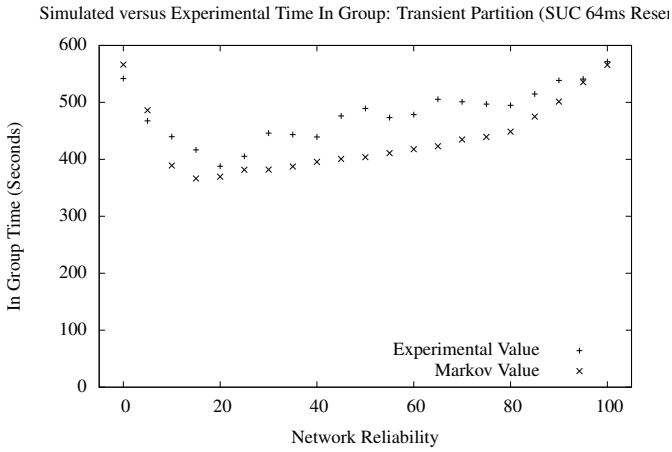

Fig. 14. Comparison of in-group time as collected from the experimental platform and the time in group from the equivalent Markov chain (64ms between resends).

Correlation was calculated from the collected data: a correlation coefficient of .8184 was found for the 128ms resend case and a factor of .8754 was found for the 64ms resend case.

8 Conclusion

This work presented a new approach for predicting the behavior of a real-time distributed system under omission failure conditions. By 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 better 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 the physical system to fail, the scheduler may need to behave in a manner that limits the number of failed migrations that can occur before group reconfigures. This work is a stepping stone towards designing a real-time schedule that manages the 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.

Acknowledgments

The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...

References

[1] R. Akella, F. 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*, Oct 2010, pp. 7–12.

[2] 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 *International Computer Software and Applications Conference, 37th Annual*, 2013.

[3] H. Garcia-Molina, "Elections in a distributed computing system," *Computers, IEEE Transactions on*, vol. C-31, no. 1, pp. 48 –59, January 1982.

[4] 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*, vol. 2, Jan 2009, pp. 106–110.

[5] Q. Dong and D. 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*, June 2009, pp. 1–9.

[6] 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*, vol. 1, April 2003, pp. 261–272 vol.1.

[7] N. Mohammed, H. Otrok, L. 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*, vol. 8, no. 1, pp. 89–103, Jan 2011.

[8] K. Birman and R. R. Los Alamitos, CA 90720-1264: IEEE Computer Society Press, 1994.

[9] R. V. Renesse, T. M. Hickey, and K. P. Birman, "Design and performance of horus: A lightweight group communications system," Cornell, Tech. Rep., 1994.

[10] L. Moser, P. 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*, vol. 39, pp. 54–63, 1996.

[11] 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*, 1992, pp. 76–84.

[12] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton, "The spread toolkit: Architecture and performance," Johns Hopkins University, Tech. Rep., 2004.

[13] 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*, 2013, pp. 180–188.

[14] 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*. Springer Berlin Heidelberg, 2013, pp. 59–70.

[15] B. J. Choi, H. Liang, X. Shen, and W. Zhuang, "Dcs: Distributed asynchronous clock synchronization in delay tolerant networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 3, pp. 491–504, March 2012.

[16] A. P. S. Meliopoulos, G. 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, pp. 8 pp.–.

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

[18] R. Sahner and K. Trivedi, "Sharpe: a modeler's toolkit," in *Computer Performance and Dependability Symposium, 1996., Proceedings of IEEE International*, Sep 1996, pp. 58–.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.