© 2000, HCS Research Lab. All Rights Reserved.

Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems

Mark W. Burns, Alan D. George, and Brad A. Wallace

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
P.O.Box 116200, Gainesville, FL 32611-6200

Abstract — Three protocols for gossip-based failure detection services in large-scale heterogeneous clusters are analyzed and compared. The basic gossip protocol provides a means by which failures can be detected in large distributed systems in an asynchronous manner without the limits associated with reliable multicasting for group communications. The hierarchical protocol leverages the underlying network topology to achieve faster failure detection. In addition to studying the effectiveness and efficiency of these two agreement protocols, we propose a third protocol that extends the hierarchical approach by piggybacking gossip information on application-generated messages. The protocols are simulated and evaluated with a fault-injection model for scalable distributed systems comprised of clusters of workstations connected by high-performance networks, such as the CPlant machine at Sandia National Laboratories. The model supports permanent and transient node and link failures, with rates specified at simulation time, for processors functioning in a fail-silent fashion. Through high-fidelity, CAD-based modeling and simulation, we demonstrate the strengths and weaknesses of each approach in terms of agreement time, number of gossips, and overall scalability.

Index Terms — Cluster computing, distributed systems, failure detection, fault tolerance, gossip protocol, Myrinet.

1 INTRODUCTION

Advances in microprocessors, networking, and distributed computing tools have led to an increased interest in cluster computing. A cluster is a collection of interconnected computers that can be programmed or managed as a single, unified computing resource. Clusters achieve high availability in a less expensive and more scalable manner than conventional fault-tolerant multiprocessors or multicomputers. Fault detection, location, and recovery mechanisms, coupled with the ability to create clusters from off-the-shelf technology, have made clusters very attractive in today's mass-market of high-availability computing systems [11].

To implement fault-tolerance in high-performance computing systems, low-level communication services, which can be implemented in hardware or software, are needed to keep overhead and complexity at a minimum. A combination of two services is considered in this paper: heartbeat mechanisms and information gathering. Heartbeats provide liveliness checks from one node to another through the use of periodic information exchanges. To effectively communicate liveliness information between nodes, mapping and information gathering mechanisms must be available. These mechanisms provide a global system map of correctly operating nodes; this information is used to efficiently route liveliness information and locate failures. Higher-level services can leverage these tools for applications such as load balancing, failure recovery, process distribution, leader election, and synchronization.

In a large distributed system it is necessary to detect failures in an asynchronous manner because network delays may be arbitrarily long. A failure detector must be able to differentiate between a dead process and a slow process that is alive. Processes and processors communicate amongst themselves to achieve consensus — the correct, timely agreement of the present state of all nodes — about the system state. Agreement may be achieved through one of several methods, most notably group communication and gossiping. Both approaches relax the constraints of absolute consensus, otherwise the implementation of these services may be extremely complex. Group communication is based on an approach where processes or processors use a reliable broadcast (or multicast) to communicate their state to other members such that the entire group agrees on the system state. With a gossip protocol, nodes individually gossip their state, and the states of other nodes, to one another to eventually achieve a unified view of the system.

1.1 Related Work

A popular method of detecting failures in a large distributed environment is group communication. A group communication service communicates with group members and processes in a manner that is independent of physical location or network delay time. Several software toolkits are available to support the construction of reliable distributed systems with group communication mechanisms. Some packages are ISIS [1], Horus [16], [17], and Transis [6], [7]. These toolkits provide a collection of high-level primitives for creating and managing process groups and implementing group-based software.

A group membership protocol manages the formation and maintenance of a set of processes called a group. A process group is a collection of programs that adapt transparently to failures and recoveries by communicating amongst themselves and exchanging information [1]. A process may be removed from a group because it failed, voluntarily requested to leave, or was forcibly expelled by other members of the group. New or repaired processes may also join a group. A group membership protocol must manage such dynamic changes in a coherent manner [5].

A major design decision must be made in the implementation of a group membership service — how to support large partitions of members within a group. One method, primary-partition, continues group membership only when a majority of the nodes remain; nodes excluded from the majority are forcibly expelled from the group. In contrast to primary-partition, partitionable services allow processes to disagree on the current membership of the group (i.e., several different views of the group may evolve concurrently and independently from each other) [5]. However, group membership services supporting partitions are notoriously difficult to achieve [9].

ISIS and Horus use a collection of information gathering processes to provide the system with functions to implement reliable distributed software. Both systems are designed with a layered approach of low-level failure detection and high-level functions for group membership. Horus is a redesigned, more efficient version of ISIS [16].

Because ISIS is designed for distributed systems, process communication is handled asynchronously. Each process relies on a low-level detector to report failures to the higher level ISIS primitives. ISIS assumes a fail-silent model as opposed to a Byzantine failure model — processes are assumed to fail by halting rather than sending incorrect or intentionally misleading messages. ISIS uses an agreement protocol to maintain a system membership list; only processes included in the list are permitted to participate in the system, and non-responsive processes are dropped. ISIS replicates knowledge of membership among all members of the group rather than implementing a membership server, which represents a possible single point of failure [1].

The ISIS system assumes a "perfect" failure detector. Low-level heartbeats are used to detect process crashes, however these must be set conservatively because premature time-outs are costly and can lead to system shutdown [4].

ISIS and Horus provide applications with a transport-level multicast communication service. Similarly, the Transis system employs a broadcast protocol, Trans, for process communication. Trans is based on a combination of positive and negative acknowledgements that are piggybacked on messages originated by a process. All communications are broadcast to all members. Acknowledgements are used as a tracking mechanism to ensure that all members receive all messages. Trans also assumes a fail-silent process model. Because of link failures, messages can be lost or delivered only to some processes. The performance of this protocol is diminished as the number of failed messages is increased [8].

Chandra et al. [5] have shown that ideal group membership involving strict consensus in asynchronous systems is not theoretically possible. Group membership also becomes inefficient as the number of processors or processes in a group increases. Although many systems have been based on group communication services, a gossip-style protocol to share liveliness information is a more scalable method. This approach to failure detection has recently been designed as a result of research with the ISIS, Horus, and Transis systems. The gossip protocol has been demonstrated analytically as an effective protocol for failure detection in distributed systems [18].

Heartbeat messages are the foundation of the gossip protocol. Each member maintains a list containing each known member, the member's address, and a heartbeat counter for timeout information. Periodically, each member gossips its membership list to one other member in the system. With this mechanism all members eventually detect process crashes. A gossip-style protocol is better suited than a group communication service for systems that do not readily support broadcast or multicast. As a result of point-to-point messages, the gossip protocol has little communication overhead. However, consensus is achieved less rapidly than with a broadcast or multicast approach.

1.2 Problem Approach

In this paper, three gossip-style protocols for large-scale, heterogeneous, distributed systems are analyzed using high-fidelity modeling and simulation. There are many applications for a high-performance, low-level, fault-tolerant service using failure detection such as a gossip-style protocol. A typical application would be fault tolerance for a system such as the Computational

Plant [13] (CPlant) at Sandia National Laboratories. The CPlant system architecture specifies a scalable, heterogeneous, massively parallel environment constructed from commodity workstations and high-performance networks. Because of the size and distributed nature of such systems, low-level services for failure detection are a necessity to serve the fault-tolerance needs of higher-level services and applications and achieve reliable and effective system operation.

The three protocols are each implemented on a fault-injection model of the CPlant system. The model is built with the Block Oriented Network Simulator (BONeS) [15], a commercial CAD tool from Cadence Design Systems. BONeS provides a graphical interface for high-fidelity system modeling, an interactive simulator for debugging, a background simulator, and post-processing data tools for system evaluation.

The fault-injection model is structured as a distributed system of multiple interconnected 16-node clusters. The clusters are connected by high-fidelity BONeS models of the Myrinet interconnection network (ICN) [2], [10], [19] developed in support of this project. The model supports permanent and transient node and link failures. Using fault injection, the gossip protocols are evaluated as a scalable solution for failure detection in a high-performance, large-scale cluster environment.

The remainder of the paper is organized as follows. In Section 2 the gossip protocols, as implemented in the model, are discussed. The structure of the fault model and coverage area is detailed in Section 3. The experiments performed and fault-injection methods used are explained in Section 4. In Section 5 the performance results are presented. Finally in Section 6, conclusions from this research are presented, and future directions are enumerated.

2 GOSSIP PROTOCOL MODELS

A simple failure detection mechanism can be implemented using a gossip protocol; each member periodically forwards information to a randomly chosen node to eventually detect nodes that have failed. The modeling, simulation, and analysis of the clustered systems are conducted using three variations of the gossip protocol — basic, hierarchical, and piggyback. The basic and hierarchical gossip protocols are implemented as defined by Van Renesse, Minsky, and Hayden [18]. The basic protocol operates without knowledge of the underlying network, while the hierarchical protocol leverages knowledge of the network structure to send a majority of gossips to local nodes. The hierarchical gossip protocol improves the performance of the basic protocol, especially in the case of network partitions. The piggybacking of gossip messages further increases the performance of failure detection by appending gossip information to application-generated messages.

Each member implementing the gossip protocol maintains a list containing each known member's address and heartbeat time. Every T_{gossip} seconds each node updates its own heartbeat time and sends its gossip list to another member randomly chosen from the list. For arriving gossip messages, the receiving node merges the received gossip list with its current gossip list and retains the latest heartbeat time for each node. A node is suspected to have failed after a period of $T_{cleanup}$ seconds has passed without receiving an updated heartbeat time, in which case the member is discarded from the gossip list.

The hierarchical gossip protocol uses an additional parameter to specify the relative percentage of gossips within and between clusters. The local gossip factor, L, improves the basic protocol's performance and scalability by considering the topology of the network; by reducing inter-cluster gossips, redundant information between clusters can be avoided while still achieving timely agreement.

The piggyback gossip protocol decreases the inter-gossip interval and/or decreases the network bandwidth required for gossip messages by piggybacking gossips on application-generated messages whenever possible. The tradeoff between gossip interval and bandwidth is controlled by a parameter that varies the rate at which messages are piggybacked. The piggyback factor, P, specifies a period within each T_{gossip} when messages cannot be piggybacked on application data, where $0 \leq P \leq 1$. Gossip messages are sent in the interval between $T_{message} = T_{gossip}*P$ and T_{gossip} seconds, where T_{gossip} and $T_{message}$ are measured relative to the most recent transmission of gossip information and $T_{message} \leq T_{gossip}$. Fig. 1 shows the intervals in which gossip information can and cannot be piggybacked on application-generated messages.

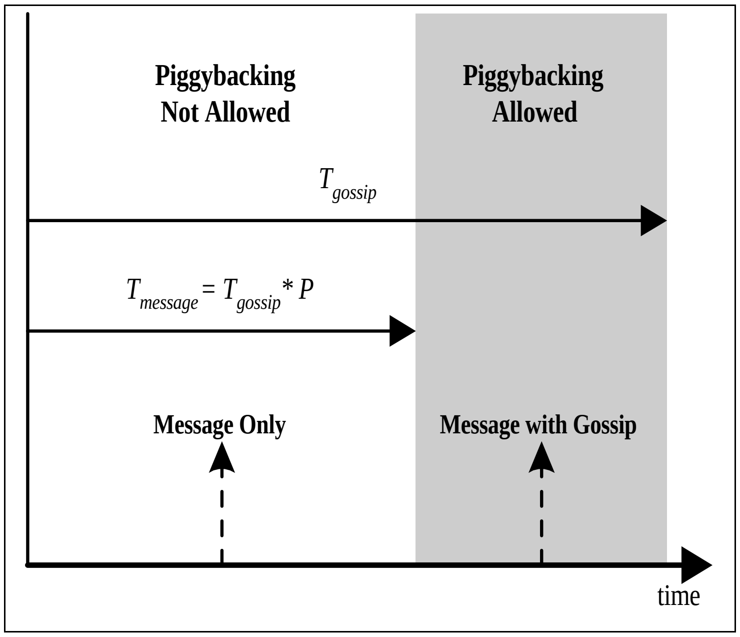

Fig. 1. Interval in which piggybacking is allowed; data messages transmitted in the shaded area will be piggybacked with gossip information.

The gossip protocol models assume the system begins at steady state; all live nodes are included in the gossip list on start-up. If this assumption were not made, each member could broadcast its list in order to be located initially. If a network does not support broadcasts, one or more gossip servers could be added to the system [18]. A gossip server is an additional member with a well-known address that is likely to be available even in the case of network partitioning.

3 SYSTEM ARCHITECTURE

The three gossip protocol variants are simulated on a multiple-cluster, distributed computing system model. Each cluster is comprised of 16 computing nodes connected by a local Myrinet network. Clusters are interconnected using a separate global Myrinet network. Fig. 2 displays the configuration of the cluster architecture discussed in this paper.

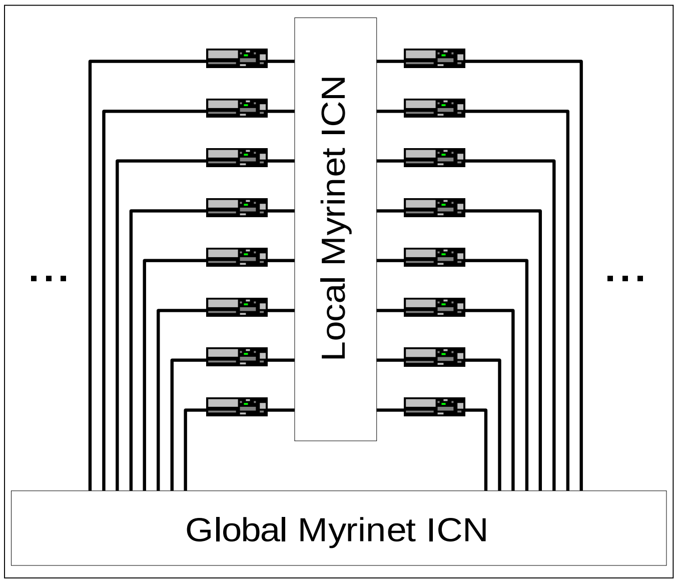

Fig. 2. Architecture for a single computing cluster with sixteen processing nodes; this configuration resembles CPlant at Sandia National Laboratories.

Each processor in a cluster has access to both a local and a global Myrinet interconnection network. Each node may route messages to either network depending on the message destination. The ports are monitored using a watchdog timer; if an error-free message has not been received at a port within a specified time, the port is declared inoperable. In the event of port failures, messages are re-routed to the remaining output port (if possible). For local messages, the global interconnection network can be used when the local port is down; it is likely that latency to deliver a local message in this manner will increase due to contention.

When a global port is down, messages are routed locally to the node with the next highest address in the cluster (using modulo-16 arithmetic). The node receiving the message determines that the final address is not local and attempts to deliver the message to its global destination. To avoid non-locally destined messages in infinite loops due to multiple global port failures, it is assumed that at least one node in a cluster will have an operational global port at any given time. Also, if a message arrives at a port that was previously declared inoperable, the watchdog timer is reset and the port resumes normal operation.

The block diagram of a single computing node is shown in Fig. 3. The processor, or set of processors, is modeled as a Poisson traffic generator with a variable inter-message rate, message size, and destination range. Processor-generated messages are created such that a specific proportion of the messages are destined for a random local node (within the same cluster) while the remaining messages are sent to a non-local node. The processor parameters allow a variety of traffic patterns to represent access patterns for a number of processing algorithms.

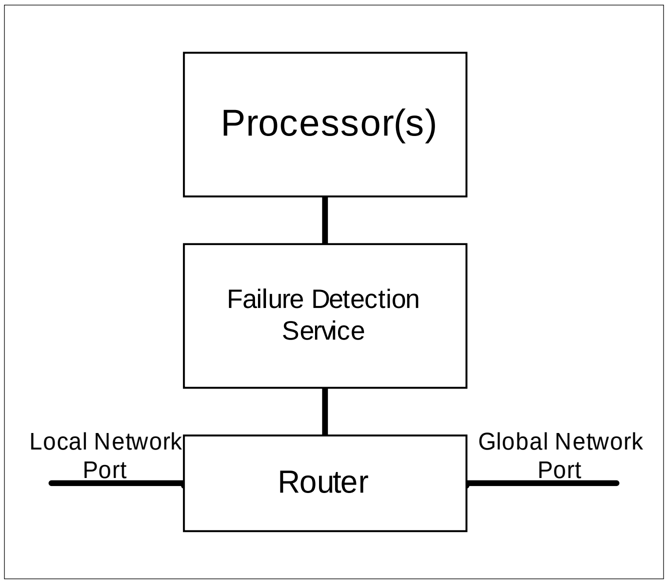

Fig. 3. Internal block diagram of a computing node.

In addition to support for the system-wide failure detection service with gossiping, the failure detection unit in each node is used to determine internal component failures. The local processor(s) and router are tested using sanity checks, and the entire node is disabled if a permanent fault is detected.

All of the Myrinet interconnection networks are composed of 8x8 crossbar switches. Each local Myrinet ICN achieves 16x16 ports with three 8x8 switches connected in a triangular fashion as depicted in Fig. 4. Using this switch architecture, local messages pass through either one or two switches before reaching a destination. Similarly, the global Myrinet ICN provides connectivity between all nodes using a hierarchy of 16x16 switches. The switch configuration varies with the number of connected clusters; global messages traverse a variable number of switches before reaching a destination.

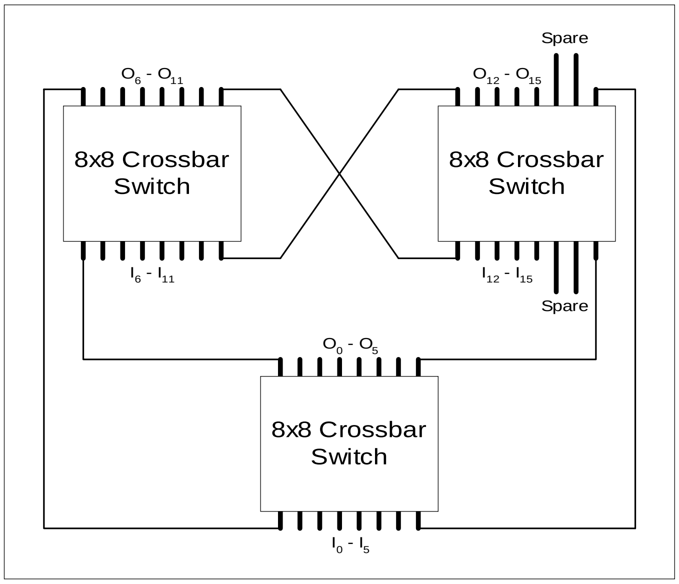

Fig. 4. Myrinet 16x16 switch topology; the global ICN uses the spare ports to provide switch-to-switch connectivity.

BONeS block diagrams for the fault-injection model are displayed in Fig. 5 and Fig. 6. Fig. 5 shows the system-level view of a 96-node system. A single cluster is displayed in Fig. 6 with two Myrinet switches for local and global connectivity.

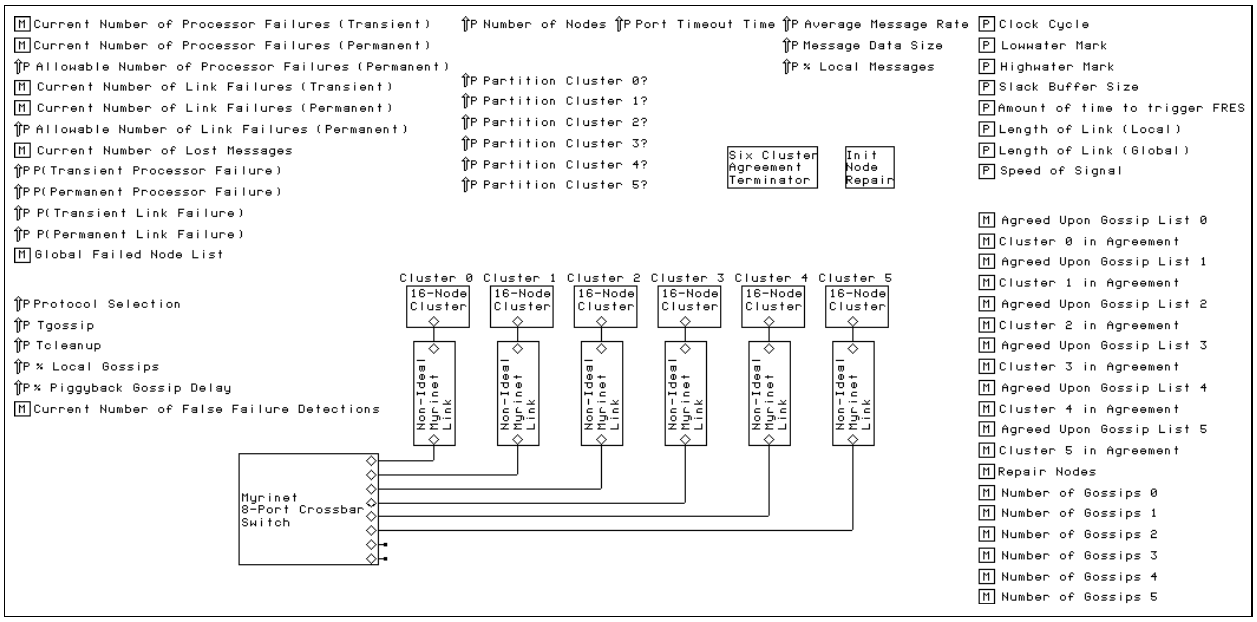

Fig. 5. System-level block diagram of the BONeS fault-injection model.

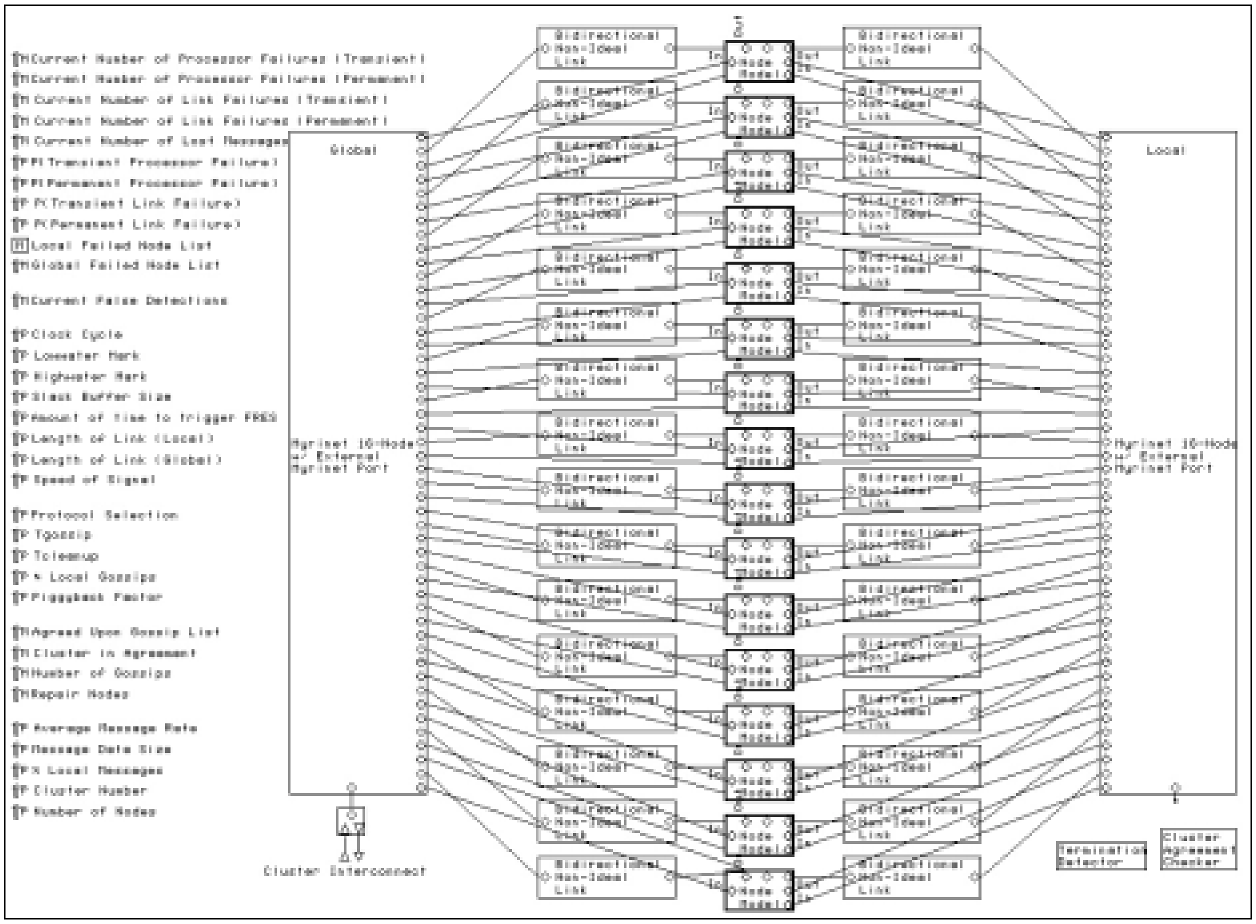

Fig. 6. Single 16-node cluster model in BONeS.

4 FAULT-INJECTION MODEL

The gossip protocol variations are simulated using a fault-injection model for the cluster-based distributed system discussed in Section 3. Because the gossip protocols are evaluated on a large, integrated system model, fault injection is done at the functional level — faults can be injected in the processors, failure detection components, or links connecting nodes to switches [12]. Myrinet switch failures are modeled with link failures between the switch and other switches or nodes. The processor network ports are assumed to be fault-free.

The system fault model is based on the successful communication of messages. Message corruption can occur in a processor, in a failure detection component, or on a communication link, and corrupted messages are discarded upon receipt by the destination node. Messages can be permanently lost by processor failures or by inoperative communication channels. Processors operate in fail-silent mode [3], [14]. Practically, fail-silent processor behavior can be approximated with checksums and other sanity checks on messages.

Permanent and transient failure rates are specified by system parameters. Processors are configured with two parameters: a probability of permanent failure and a probability of transient failure. Permanent failures cause a processor to stop functioning while transient failures corrupt a single message. Communication links are configured with similar parameters for permanent and transient failures. Because each link has a specified probability of transient failure, messages that traverse several links suffer from a higher probability of corruption than if a probability of message loss was simply specified. This distinction is important for large systems where messages must often travel through several ICN stages.

The fault-injection model also supports network partitions. Due to the architecture of crossbar switches, the model isolates a cluster from the rest of the globally connected clusters in the case of a network partition. A system parameter is used to specify the probability of a network partition occurring in one or more of the clusters.

5 Performance Results

Several configurations are simulated with the fault-injection model to obtain performance measurements for the gossip protocol parameters. The Myrinet network models have been validated with both analytical and experimental methods, and the basic and hierarchical gossip protocol models have been verified against the analytical results of Van Renesse et al. [18].

Initially, the network and traffic settings are specified for the system. Then, the impact of variations in the T_{gossip} and $T_{cleanup}$ parameters on average agreement time is determined experimentally. In a similar fashion, the impact of the local gossip factor, L, and the piggyback factor, P, are analyzed for the hierarchical and piggybacked systems, respectively. After selecting representative values for these parameters, performance results for each protocol variation with varying probabilities of transient link failure are examined. Two metrics are used to compare the fault-injection results for each protocol — agreement time and the number of gossip messages required. The agreement time is measured as the amount of time for all nodes to reach agreement on the current state of all processors in the system (alive or dead). Finally, the performance of the protocols is observed after the occurrence of one or more network partitions.

Each simulation is initialized such that one node has failed in the past and the failed node is no longer listed in the gossip list of any active node. At the start of simulation, the failed node is repaired and becomes active in the system. Because the gossip protocol is based on randomly generated destinations, each fault-injection simulation is executed several times, and the figures represent the average performance of the failure detection algorithm.

5.1 Network and Traffic Settings

The configuration for the high-fidelity BONeS models of Myrinet used throughout the simulations is shown in Table 1. These parameters configure the models to operate in a comparable manner to the Myrinet testbed in our research laboratory with which they have been validated. The application-generated message traffic is configured to represent a typical parallel algorithm. The offered load is set to 20MB/s per node. The message size is fixed at 512 bytes to provide small message latencies across the global ICN. A representative traffic mixture of 70% intra-cluster and 30% inter-cluster is employed.

Table 1. Myrinet model parameters for all simulations.

Term	Definition	Value
Send Rate	Myrinet character output rate	160 MB/s
FRES Time	Forward reset for failures	*disabled*
Slack Buffer Size	Queue for flow control	96 flits
Lowwater Mark	Specifies low load; GO sent	40 flits
Highwater Mark	Specifies high load; STOP sent	56 flits
Length of Local Link	Intra-cluster distance	1 m
Length of Global Link	Inter-cluster distance	5 m
Speed of Signal	Propagation rate (0.6 * *c*)	1.8E8 m/s

5.2 Protocol Analysis

Simulation results with the basic gossip protocol on a 96-node model are analyzed, where ideal links are assumed. Fig. 7 shows the time to reach agreement for various gossiping intervals with a 96-node system. In the figure, T_{gossip} is varied while the message size and the proportion of intra-cluster messages is fixed at 512 bytes and 70%, respectively. $T_{cleanup}$ is set at 10ms. For small values of T_{gossip}, the time to reach agreement increases due to the congestion imposed by excessive gossiping on the network. For large values of T_{gossip}, the time to reach agreement is large because the gossip list information is not propagating from node to node at an efficient rate. If T_{gossip} is large enough to avoid network saturation (i.e., $T_{gossip} \geq 0.02$ms in this case), then the agreement time scales with T_{gossip}. However, although a minimum agreement time may be desired, the smaller gossiping interval necessary to achieve this agreement time will require a greater amount of network bandwidth. Therefore, a designer must carefully select the T_{gossip} value that achieves the best balance between agreement time and burden on the network (e.g., a value of T_{gossip} = 0.1ms is used in subsequent simulations to balance these factors). Of course, changes in the system size or application traffic patterns could alter the overall network load and adversely affect the minimum agreement time that can be attained.

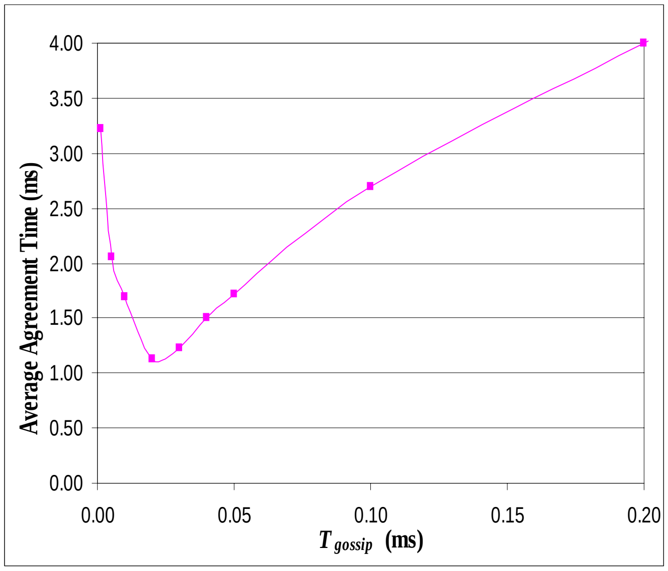

Fig. 7. Time to reach agreement for various gossiping intervals.

Fig. 8 demonstrates that the time to reach agreement also varies with the time to discard suspected failed nodes from the gossip list, $T_{cleanup}$. For small values of $T_{cleanup}$, the time to reach agreement is large due to false failure detections; a false failure detection occurs when a node is incorrectly purged from another node's gossip list even though the purged node is still alive. For small values of $T_{cleanup}$ (i.e., $T_{cleanup} < 30*T_{gossip}$ for the 96-node system), agreement becomes unreachable because each node falsely detects all other nodes as failed. For larger values of $T_{cleanup}$, false failure detections are minimized, and the time to reach agreement becomes relatively constant. For all remaining simulations, $T_{cleanup}$ is set to 10ms (i.e., $T_{cleanup}$ =

100*T_{gossip}) so that agreement time is minimized as a result of no false failure detections. For practical applications of the gossip protocol, $T_{cleanup}$ would be adjusted to minimize two conflicting goals: the number of false failure detections and the failure detection latency seen by the application.

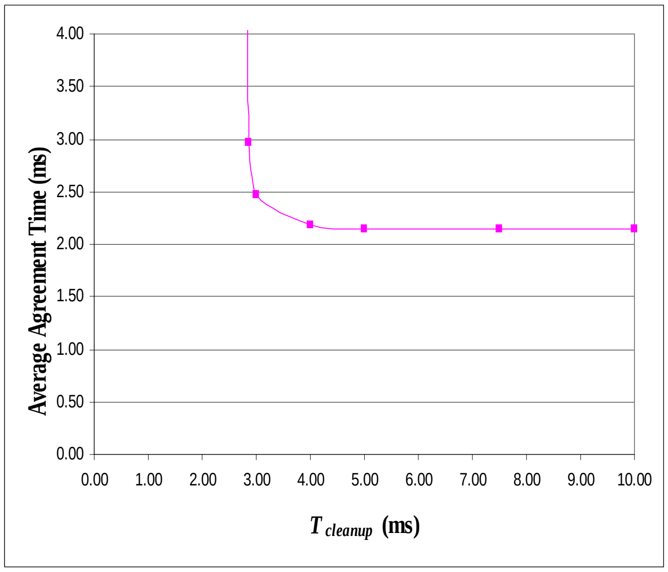

Fig. 8. Agreement time versus $T_{cleanup}$ for the 96-node system; T_{gossip} = 0.1ms.

Fig. 9 shows the agreement time for the hierarchical protocol as the local gossip factor is varied. The optimum value for L is approximately 0.9, which yields the minimum agreement time.

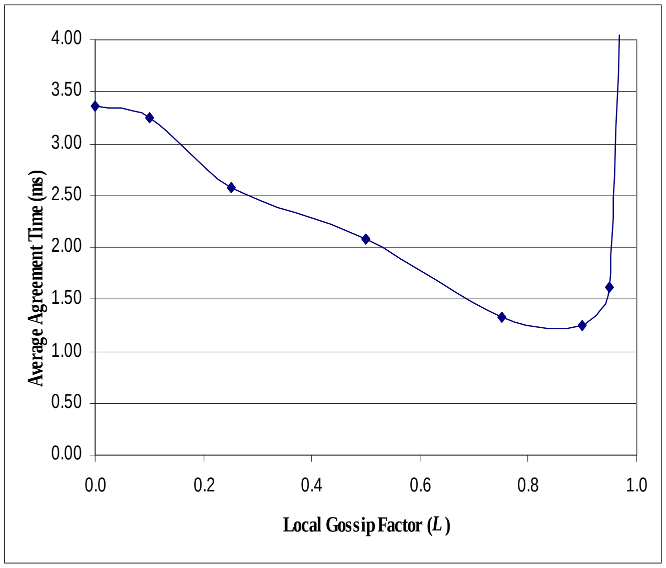

Fig. 9. Agreement time for varying local gossip factors for the hierarchical 96-node system; T_{gossip} = 0.1ms and $T_{cleanup}$ = 10ms.

For a 96-node system, the agreement time using the piggyback gossip protocol with the piggyback factor varied is shown in Fig. 10. Ideally, the optimal piggyback factor would minimize the agreement time. However, as P is decreased, more bandwidth is consumed by gossip messages. For large systems, gossip messages contribute to even higher network loads; therefore the largest P minimizes the required bandwidth. The optimum factor is selected by minimizing both agreement time and required bandwidth; in this study, a piggyback factor of 0.75 (i.e., $T_{message} = 0.75*T_{gossip}$) is used in the simulation experiments.

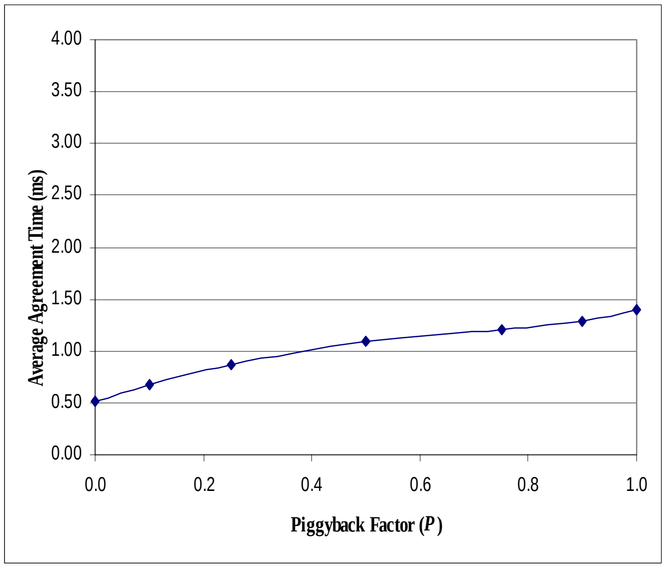

Fig. 10. Agreement time for varying piggyback factors; T_{gossip} = 0.1ms and $T_{cleanup}$ = 10ms.

5.3 Protocol Comparisons

Fig. 11a and Fig. 11b show the agreement time and the number of gossips, respectively, for failure detection of one permanent processor failure using the basic gossip protocol. Systems with 16 to 96 processors are simulated, and transient link failure probabilities, *p*, of 0, 0.05, and 0.1 are shown. The number of gossips cited represents the total number of uncorrupted gossip messages received to reach agreement; corrupted messages are discarded upon receipt and not counted in the total. Agreement time increases for higher link failure rates because more messages must be sent to achieve the number of gossip messages required for agreement. In Fig. 11b, the average number of gossip messages for the different transient link failure probabilities is approximately equal for each node size; variations in the number of messages are a result of the randomness of the protocol.

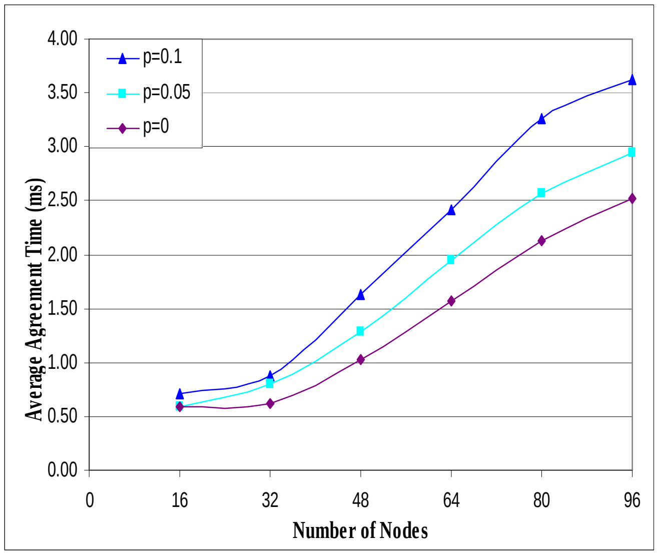

Fig. 11a. Agreement times for the basic gossip protocol model; T_{gossip} = 0.1ms and $T_{cleanup}$ = 10ms.

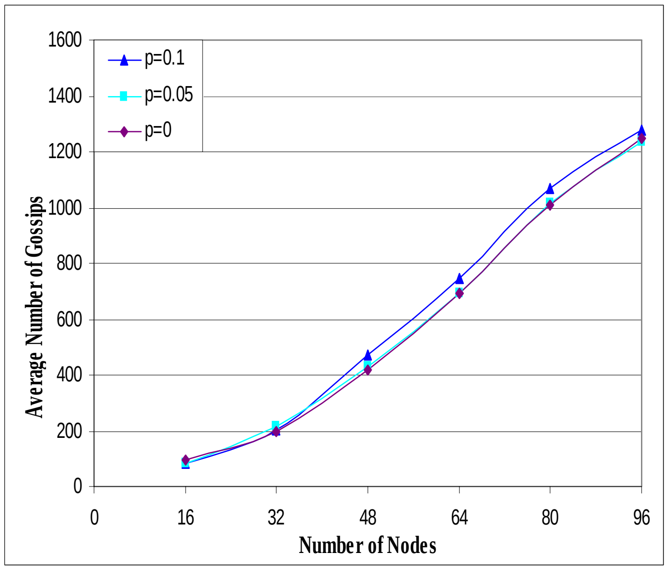

Fig. 11b. Number of gossip messages required to reach agreement for the basic gossip protocol model; T_{gossip} = 0.1ms and $T_{cleanup}$ = 10ms.

The performance results for the hierarchical gossip protocol model are shown in Fig. 12a and Fig. 12b. Notice that the time to reach agreement is much improved over the basic gossip protocol model as the number of nodes is increased (e.g., 1.54ms versus 2.53ms for 96 nodes with p = 0). The average number of gossip messages needed to reach agreement is slightly higher than that of the basic gossip protocol (e.g., 1423 messages for the hierarchical protocol versus 1251 messages for the basic protocol using the 96-node system with p = 0). Since the hierarchical protocol sends 90% of its gossips locally, more total gossip messages are required to reach agreement across the system. The hierarchical protocol achieves agreement more quickly than the basic protocol at the expense of bandwidth consumed in the local cluster network. Using the average slopes of the curves as a measure of relative scalability with respect to growth in system size, it becomes evident that agreement times with the hierarchical protocol increase at only 59% the rate of the basic protocol.

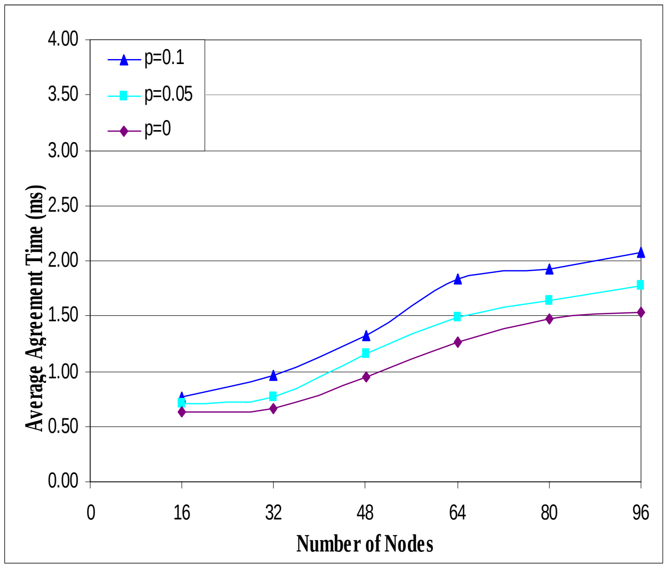

Fig. 12a. Agreement times for the hierarchical protocol model; T_{gossip} = 0.1ms, $T_{cleanup}$ = 10ms, and L = 0.9.

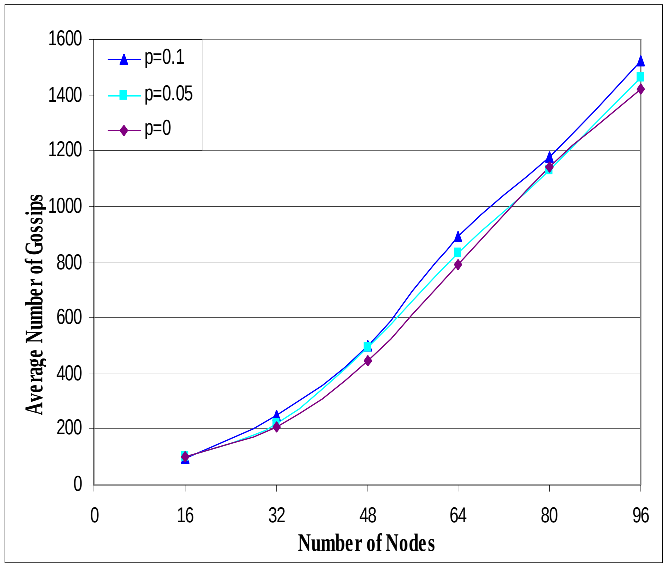

Fig. 12b. Number of gossip messages required to reach agreement for the hierarchical gossip protocol model; T_{gossip} = 0.1ms, $T_{cleanup}$ = 10ms, and L = 0.9.

Fig. 13a and Fig. 13b show the average agreement times and average number of gossip messages needed to achieve agreement for the piggyback gossip protocol. The time to reach agreement is lower for the piggyback case than the other protocol variations (e.g., 1.21ms for the piggyback protocol versus 1.54ms for the hierarchical protocol on a 96-node system with p = 0). Agreement times with the piggyback gossip protocol increase with system size at only 45% the rate of the basic protocol and 77% the rate of the hierarchical protocol. The number of gossip messages is comparable to the basic protocol however, as illustrated by the closer proximity of the curves in the figures, the performance of the piggyback protocol is affected less by transient link failures.

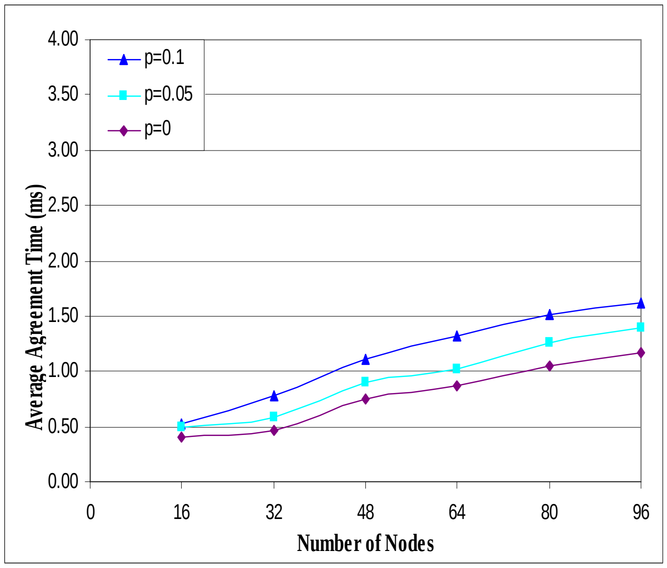

Fig. 13a. Agreement times for the piggyback gossip protocol model; T_{gossip} = 0.1ms, $T_{cleanup}$ = 10ms, and P = 0.75.

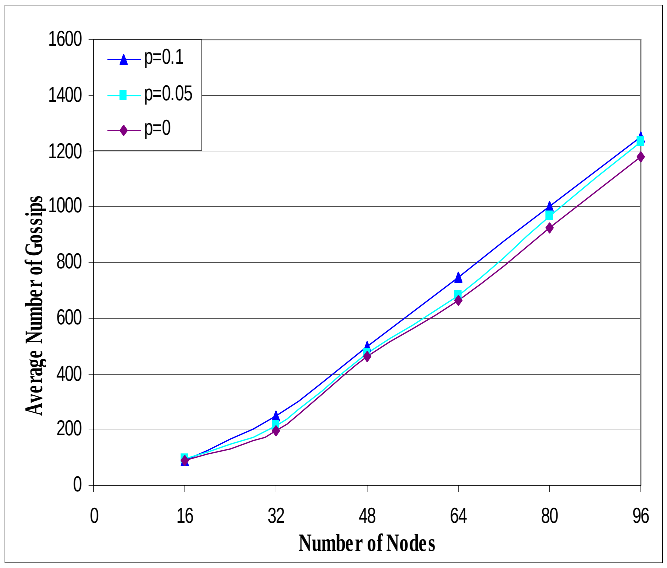

Fig. 13b. Number of gossip messages required to reach agreement for the piggyback gossip protocol model; T_{gossip} = 0.1ms, $T_{cleanup}$ = 10ms, and P = 0.75.

5.4 Partition Performance

The ability of each protocol variation to handle network partitions in a 96-node system is analyzed assuming no transient link failures. Based on the previous results in Figs. 11, 12, and 13, transient link failures have a minor impact on the rate of growth in agreement time. As mentioned in Section 4, a network partition occurs when a cluster is permanently disconnected from the global ICN. Fig. 14 shows the agreement times for network partitions of one, two, three, four, five, and six clusters. In each case, the hierarchical and piggyback gossip protocols achieve agreement in approximately half the time as the basic protocol. As a general trend, agreement time for the hierarchical and piggyback protocols decreases as the network becomes more partitioned. This trend occurs because the largest group of nodes required to agree is smaller as more clusters are partitioned; both variations transmit a majority of their gossips locally, therefore a majority of their gossips are not discarded. For the cases of five and six cluster partitions, the hierarchical protocol exhibits a lower agreement time than the piggyback protocol. This result is expected because the hierarchical protocol sends 90% of its gossips locally; the piggyback protocol sends a majority of the gossips attached to application-generated messages, which send 70% of their gossips locally. The larger local gossip factor aids the performance of the hierarchical protocol for catastrophic network partitions. The basic protocol does not benefit from smaller group sizes because nodes are selected at random and a larger percentage of gossip messages are sent to inaccessible destinations. With the basic protocol, the agreement time initially decreases as the number of partitions increases because the largest group of nodes that must agree becomes smaller. However, as the number of partitions increases further, the percentage of gossips sent to unreachable nodes increases. The basic protocol becomes less efficient as more partitions occur, and therefore the agreement time increases.

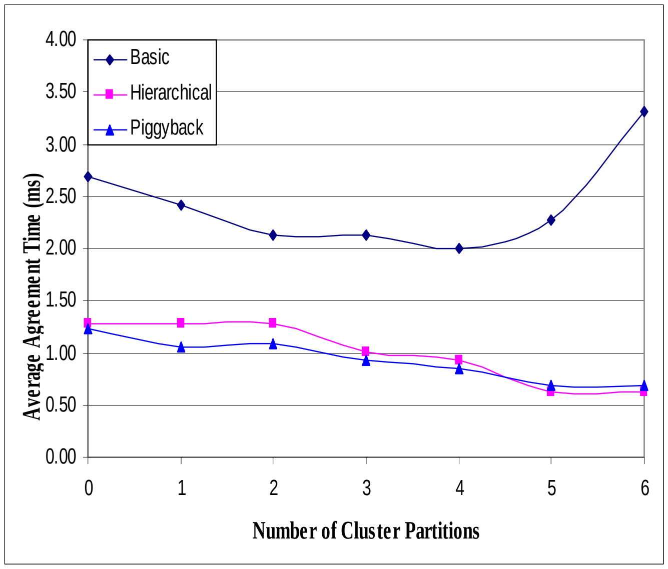

Fig. 14. Protocol agreement time for network partitions; T_{gossip} = 0.1ms, $T_{cleanup}$ = 10ms, L = 0.9, and P = 0.75.

6 Conclusions

In this paper, we present a fault-injection evaluation of a failure detection service for high-performance clusters. We analyze two existing gossip protocol variations and propose and evaluate a new approach leveraging application-generated messages for decreased failure detection time.

The results of the simulative analyses indicate that a gossip-style protocol may be an effective means of failure detection for a high-performance, cluster-based system. The basic gossip protocol yields the slowest agreement time and requires a large number of gossip messages to be exchanged, but is the least complex and only requires configuration of two parameters. A significant reduction in agreement time can be achieved with the hierarchical protocol, however more parameter configuration is necessary to optimize the protocol for the topology of the interconnection network. Compared to the basic protocol, the hierarchical protocol is more scalable with respect to system size (e.g., achieving agreement times with respect to system size that increase at 59% the rate of the basic protocol in our experiments). The piggyback protocol is the most sophisticated of the three protocols analyzed, where monitoring of application-generated traffic patterns is required to effectively attach gossip messages. The piggyback protocol has superior performance and scalability (e.g., achieving agreement times with respect to system size that increase at 45% the rate of the basic protocol) compared to the other two protocols at the expense of network bandwidth, which can be dynamically controlled by the piggyback parameters.

Areas of future research include parameter optimizations for additional system configurations and failure scenarios as well as analysis of promising hybrids. One such alternative is a combination of both group and gossip communication, where each cluster in a distributed system uses a broadcast-based protocol to share state information within the cluster, and a gossip protocol supports the propagation of state information between clusters throughout the system. Another alternative involves the use of a "buddy system" where nodes are logically paired together for low-level failure detection, and the logical pairs work in concert with a group, gossip, or hybrid detection scheme to achieve system-wide failure detection.

ACKNOWLEDGEMENTS

The authors would like to thank Robbert Van Renesse and Werner Vogels at Cornell University for their assistance in the initial research that preceded this project, and Roger VanLoon in our lab for his many contributions with the Myrinet model. This research was supported in part by Sandia National Laboratories and the National Security Agency.

REFERENCES

[1] K. Birman, "The Process Group Approach to Reliable Distributed Computing," *Communications of the ACM*, vol. 36, no. 12, pp. 37-53, December 1993.

[2] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro,* vol. 15, no. 1, pp. 26-36, February 1995.

[3] F. Brasileiro, P. Ezhilchelvan, S. Shrivastava, N. Speirs, and S. Tao, "Implementing Fail-Silent Nodes for Distributed Systems," *IEEE Transactions on Computers*, vol. 45, no. 11, pp. 1226-1238, November 1996.

[4] T. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *Journal of the ACM*, vol. 43, no. 4, pp. 685-722, July 1996.

[5] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, "Impossibility of Group Membership in Asynchronous Systems," *In Proceedings of the 15th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Philadelphia, Pennsylvania, pp. 322-330, May 1996.

[6] D. Dolev and D. Malki, "The Design of the Transis System," *Lecture Notes in Computer Science*, no. 938, Springer-Verlag, pp. 83-98, 1995.

[7] D. Dolev and D. Malki, "The Transis Approach to High Availability Cluster Communication," Technical Report 94-14, Computer Science Institute, Hebrew University, Jerusalem, Israel, 1995.

[8] P. Melliar-Smith, L. Moser, and V. Agrawala, "Broadcast Protocols for Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1, pp. 17-25, January 1990.

[9] S. Mullender, *Distributed Systems*, New York: ACM Press, 1989.

[10] Myricom, "Myrinet link specification," *http://www.myri.com/scs/documentation/link/index.html*, 1995.

[11] G. Pfister, *In Search of Clusters*, 2nd Edition, Upper Saddle River, NJ: Prentice Hall 1998.

[12] D. Pradhan, *Fault-tolerant Computer System Design*, Upper Saddle River, NJ: Prentice Hall, 1996.

[13] Sandia National Laboratories, "Computational Plant," *http://rocs-pc.ca.sandia.gov/CPlant/CPlant.html*, 1998.

[14] R. Schlichting and F. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Transactions on Computing Systems*, vol. 1, no. 3, pp. 222-238, August 1983.

[15] K. Shanmugan, V. Frost, and W. LaRue, “A Block-Oriented Network Simulator (BONeS),” *Simulation*, vol. 58, no. 2, pp. 83-94, February 1992.

[16] R. Van Renesse, T. Hickey and K. Birman, “Design and Performance of Horus: A Lightweight Group Communications System,” Technical Report 94-1442, Department of Computer Science, Cornell University, Ithaca, New York, December 1994.

[17] R. Van Renesse, K. Birman, B. Glade, K. Guo, M. Hayden, T. Hickey, D. Malki, A. Vaysburd, and W. Vogels, “Horus: A Flexible Group Communication System,” Department of Computer Science, Cornell University, Ithaca, New York, March 1993.

[18] R. Van Renesse, Y. Minsky, and M. Hayden, “A Gossip-Style Failure Detection Service,” *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, September 15-18, 1998.

[19] VITA Standards Organization, “Myrinet-on-VME Protocol Specification Draft Standard,” *http://www.vita.com/vso/draftstd/myri-vme-d05.pdf*, 1998.