

2006

# Efficient failure detection for point-to-point communication networks

Jeremy Dahlgren

Follow this and additional works at: <http://scholarworks.rit.edu/theses>

---

## Recommended Citation

Dahlgren, Jeremy, "Efficient failure detection for point-to-point communication networks" (2006). Thesis. Rochester Institute of Technology. Accessed from

This Master's Project is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact [ritscholarworks@rit.edu](mailto:ritscholarworks@rit.edu).

# Efficient Failure Detection Protocols for Point-to-Point Communication Networks

Master's Thesis

Jeremy Dahlgren  
(jad0883@cs.rit.edu)

**Chair** - Hans-Peter Bischof  
(hpb@cs.rit.edu)

**Reader** - Alan Kaminsky  
(ark@cs.rit.edu)

**Observer** - Paul Tymann  
(ptt@cs.rit.edu)

January 8, 2004

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>6</b>  |
| <b>2</b> | <b>Failure Detection Requirements and Assumptions</b>         | <b>8</b>  |
| 2.1      | Requirements . . . . .  | 8         |
| 2.1.1    | Efficiency and Completeness . . . . .                         | 8         |
| 2.1.2    | Scalable . . . . .  | 9         |
| 2.1.3    | Message Usage . . . . .                                       | 9         |
| 2.1.4    | Computational Complexity . . . . .                            | 9         |
| 2.2      | Assumptions . . . . .   | 10        |
| 2.2.1    | Network . . . . .   | 10        |
| 2.2.2    | Protocol Messages . . . . .                                   | 10        |
| 2.2.3    | Point-to-Point Communication . . . . .                        | 10        |
| 2.2.4    | Failure Detection vs. Group Membership . . . . .              | 11        |
| 2.2.5    | Group Sizes . . . . .   | 11        |
| <b>3</b> | <b>Available Protocols</b>                                    | <b>11</b> |
| 3.1      | Heartbeat . . . . .   | 11        |
| 3.1.1    | Description . . . . .   | 12        |
| 3.1.2    | Discussion . . . . .  | 14        |
| 3.2      | Ping . . . . .  | 15        |
| 3.2.1    | Description . . . . .   | 15        |
| 3.2.2    | Discussion . . . . .  | 17        |
| 3.3      | Gossip . . . . .  | 18        |
| 3.3.1    | Description . . . . .   | 19        |
| 3.3.2    | Discussion . . . . .  | 22        |
| 3.4      | Protocol Proposed by Chandra, Goldszmidt, and Gupta . . . . . | 23        |
| 3.4.1    | Description . . . . .   | 24        |
| 3.4.2    | Discussion . . . . .  | 28        |
| <b>4</b> | <b>Aggressive Protocol</b>                                    | <b>29</b> |
| 4.1      | Basis . . . . .   | 29        |
| 4.2      | Modifications . . . . .                                       | 30        |
| 4.2.1    | Round-Robin Selection . . . . .                               | 30        |
| 4.2.2    | Node Characterization . . . . .                               | 30        |
| 4.2.3    | Selection Based on Characterization . . . . .                 | 31        |
| 4.2.4    | Burst Notices . . . . .                                       | 31        |
| 4.3      | Discussion . . . . .  | 33        |

|           |  |           |
|-----------|--|-----------|
| <b>5</b>  | <b>Protocol Comparisons</b>              | <b>35</b> |
| 5.1       | Messages Used . . . . .                  | 35        |
| 5.2       | Accuracy versus Speed . . . . .          | 38        |
| 5.3       | Complexity . . . . .                     | 40        |
| 5.4       | Summary . . . . .                        | 40        |
| <b>6</b>  | <b>Protocol Implementations</b>          | <b>41</b> |
| 6.1       | Abstract Device Base Class . . . . .     | 41        |
| 6.1.1     | Communication . . . . .                  | 41        |
| 6.1.2     | Event Generation . . . . .               | 42        |
| 6.1.3     | Suspect List . . . . .                   | 43        |
| 6.1.4     | Simulation Functionality . . . . .       | 43        |
| 6.2       | Design of Protocol Subclasses . . . . .  | 43        |
| <b>7</b>  | <b>Test Framework</b>                    | <b>44</b> |
| 7.1       | Input Files . . . . .                    | 44        |
| 7.1.1     | Example Input File . . . . .             | 44        |
| 7.1.2     | Grammar . . . . .                        | 46        |
| 7.1.3     | Import Statement . . . . .               | 48        |
| 7.1.4     | Input File Semantics . . . . .           | 48        |
| 7.1.5     | Keyword and Symbol Adjustments . . . . . | 49        |
| 7.2       | Parser . . . . .                         | 50        |
| 7.3       | Tester and Setup Classes . . . . .       | 50        |
| 7.3.1     | Setup . . . . .                          | 51        |
| 7.3.2     | Tester . . . . .                         | 51        |
| <b>8</b>  | <b>Tests and Results</b>                 | <b>54</b> |
| 8.1       | Protocol Parameters . . . . .            | 55        |
| 8.2       | Group Sizes . . . . .                    | 56        |
| 8.3       | Network Reliability . . . . .            | 62        |
| 8.4       | Node Device Capabilities . . . . .       | 68        |
| <b>9</b>  | <b>Conclusions</b>                       | <b>74</b> |
| <b>10</b> | <b>Future Work</b>                       | <b>75</b> |
| <b>A</b>  | <b>Protocol Parameter Data</b>           | <b>76</b> |
| A.1       | Heartbeat Parameter Data . . . . .       | 76        |
| A.2       | Ping Parameter Data . . . . .            | 78        |
| A.3       | Gossip Parameter Data . . . . .          | 80        |
| A.4       | Random Ping Parameter Data . . . . .     | 85        |
| A.5       | Aggressive Parameter Data . . . . .      | 89        |

|          |  |            |
|----------|--|------------|
| <b>B</b> | <b>Group Size Data</b>                         | <b>93</b>  |
| B.1      | Heartbeat Group Size Data . . . . .            | 93         |
| B.2      | Ping Group Size Data . . . . .                 | 94         |
| B.3      | Gossip Group Size Data . . . . .               | 95         |
| B.4      | Random Group Size Data . . . . .               | 96         |
| B.5      | Aggressive Group Size Data . . . . .           | 96         |
| <b>C</b> | <b>Network Reliability Data</b>                | <b>97</b>  |
| C.1      | Heartbeat Network Reliability Data . . . . .   | 97         |
| C.2      | Ping Network Reliability Data . . . . .        | 99         |
| C.3      | Gossip Network Reliability Data . . . . .      | 100        |
| C.4      | Random Ping Network Reliability Data . . . . . | 102        |
| C.5      | Aggressive Network Reliability Data . . . . .  | 104        |
| <b>D</b> | <b>Device Type Data</b>                        | <b>105</b> |
| D.1      | Heartbeat Device Type Data . . . . .           | 106        |
| D.2      | Ping Device Type Data . . . . .                | 106        |
| D.3      | Gossip Device Type Data . . . . .              | 106        |
| D.4      | Random Ping Device Type Data . . . . .         | 107        |
| D.5      | Aggressive Device Type Data . . . . .          | 107        |

## Disclaimer

All of the experiments in this thesis were carried out in the Rochester Institute of Technology's Computer Science Department network of workstations. As a result factors such as workstation load, network traffic, and network delay could not be controlled. This means that the results presented in this thesis are not conclusive. A more ideal approach would have been to implement the protocols and test framework in a completely simulated environment. Doing so would provide control over all variables and yield more accurate results. The same set of tests conducted in this thesis need to be carried out in a completely simulated environment before any final conclusions can be made.

# 1 Introduction

Suppose your company uses a multi-user media conference application. Employees can participate from anywhere in the building, or even from remote locations at other offices all over the world. Since these business conferences are private, the application encrypts the data transmitted. The encryption uses a key that is generated by using contributions from all the participating members. This ensures that the key is unique to the participating members and protects against attackers. If one of the participants' process terminates unexpectedly, it is important for the application to generate a new key for the remaining group members so that an intruder can not masquerade as the failed process. If a process failure is not detected quickly it gives an attacker an opportunity to receive group messages, and given enough time, the opportunity to discover the key used for encryption. If this happens the intruder will have access to information sent between processes. This threat is why an important part of the application is how it detects the failure quickly on every host. Fast detection protects against attackers and ensures the privacy of the conference. Accuracy is also important since key generation can be an expensive operation. So the application needs to be sure that a host has really failed before executing the key generation procedure.

Consider an application that monitors a group of shared resources on many different host computers. The hosts may be located in many different remote locations. The application provides client processes with access to the resources they need. Suppose a computer that hosts one of the shared resources fails unexpectedly. The application does not want to point clients to unavailable resources. The hosts running the application need some way of detecting if one of them fails so that the list of available resources can be updated accordingly. In this case the threat to the application is the possibility of decreased performance. The usefulness of the application depends on how fast the list is updated as new resources become available or existing resources become unavailable. The accuracy of the resource list depends not only on how fast unavailable resources are removed, but also on the fact that resources that are available are not mistakenly removed from the list.

There is a common theme underlying these two examples. The applications involve processes running on many different host computers. The computers may be organized in a local area network or even a wide area network. Detecting when one of these computers fails or when the application process running on the computer terminates unexpectedly is important to both application examples. The applications need to rely on some sort of failure detection component. This component is responsible for detecting process failures quickly and accurately, as well as making this information available to the upper level application layer. The application can then react to the failures accordingly. Besides being fast and accurate, the failure detector implementation needs to make sure that failures are detected on all hosts, whether through

independent discovery or interprocess communication.

The job of the failure detection component might sound fairly simple. Just detect when a process fails, and make the detection available to the upper layer of the application. More investigation into the design and implementation of a failure detector reveals that this is not exactly the case. Trade-offs exist between some of the properties of a failure detector. An example is how accuracy may be lost with improved speed of detection. There are several properties that a failure detector needs to have to be a valuable application component. A failure detector needs to be fast, accurate, ensure that a failure is detected at every host, scalable to accommodate large numbers of processes, efficient in terms of messages used to limit network load, and lightweight so that the CPU is available for application execution.

The way a failure detector communicates detections to the upper layer of the application is through the use of a suspect list. A suspect list is a mapping of node identifiers to a time stamp of when that node was detected. Every node maintains its own suspect list. When a node detects another node as failed it adds an entry to the suspect list. This entry contains the faulty node's identifier and the current time stamp. If the upper layer of the application wants to know what nodes the failure detector suspects as failed at any given moment, it just looks at the failure detector's suspect list.

Section two of the thesis will examine the failure detection properties previously mentioned as well as some assumptions made throughout the paper. The third section will introduce the Heartbeat [11], [5], [3], [2], [9], [8], [12], [4], [1], Ping [5], [9], [12], and Gossip [13], [10], [12], failure detection protocols, as well as a Randomized Ping protocol introduced by Chandra, Goldszmidt, and Gupta in [9]. The fourth section will present a new failure detection protocol that is based on Chandra, Goldszmidt, and Gupta's protocol. The fifth section will compare the protocols and make conclusions on which protocols are best suited for different application demands. Section six will describe the implementations of the protocols themselves. Section seven will discuss the framework used to test the protocols. Section eight will present experiments done to investigate the performance of the failure detection variants as well as the results of those experiments. Some final conclusions are presented in section nine and section ten discusses areas for future work.



## 2 Failure Detection Requirements and Assumptions

### 2.1 Requirements

There are several requirements that determine the performance and usefulness of any failure detection system. Some of these properties are discussed in detail in [6], [4], and [7]. The four major requirements that this thesis focused on include efficiency and completeness, scalability, message complexity, and computational complexity. Each of these requirements should be taken into consideration when designing a failure detection protocol.

#### 2.1.1 Efficiency and Completeness

One necessary property of any failure detection protocol is completeness. The completeness property requires that every node eventually detects a failed node. Completeness is essential to the group's ability to maintain a consistent view of the nodes in the group. If some nodes do not detect a failure then it may be hard for the upper layer of the application to coordinate the removal of a node. Guaranteeing that the completeness property holds for a given protocol may require compromises to other properties of the protocol. An example of this is a protocol having to ping every node in the group to detect failures instead of having the nodes distribute the ping load. In a distributed ping environment a node may never receive notification of a node that failed that is not on its list of nodes to ping. The consequences of each node having to ping all other nodes are increases in detection times and messages sent, but each node is guaranteed to detect a failure since they ping every node.

Failure detection protocols also need to be efficient. Efficiency entails that a failed node is detected as fast as possible. Although failures need to be detected quickly, the detection mechanisms must also be accurate. A failure detection mechanism is accurate if it can correctly detect failed nodes, while minimizing the number of correct nodes being declared as failed. The strong completeness and strong accuracy properties discussed in [6] demand that *all* failures are detected by *all* nodes, and *no* non-faulty nodes are detected as failed by *any* of the other non-faulty nodes. A trade-off exists between the accuracy property and the efficiency property. Improvements in efficiency may hinder the accuracy of any detection protocol. Improving the accuracy of a protocol usually involves increasing the timeout values for most protocols, and as a direct result detection times rise. A main goal of the thesis was to see what failure detection protocols satisfy these properties.

### **2.1.2 Scalable**

Applications that depend on a failure detection protocol will require a protocol that is scalable to some degree. This means that the protocol must be able to maintain its completeness and efficiency properties as the number of nodes using the application grows. Scalability requirements may vary depending on the nature of the application. Private conference applications may expect the size of the group to be less than twenty or so members. Other applications such as distributed sensor networks may involve thousands of nodes and will need a failure detection protocol that can scale accordingly. A failure detector scales well if it can adjust to any number of group members ranging from tens of nodes to tens of thousands of nodes, without severe losses in speed or accuracy. Section 2.2 will address the scalability requirements assumed throughout this thesis.

### **2.1.3 Message Usage**

The applications that depend on failure detection protocols may be communicating through a shared medium where network traffic may be heavy at times. A goal for any failure detection protocol should be the minimization of the messages sent so the communication medium is still available for application messages as well as other network traffic. A failure detector's design must consider factors such as the frequency of messages sent as well as message size.

Failure detectors used in applications running in a wide area network will need to send messages across routers and gateways. The number of messages that the failure detectors send across these routes is more of an issue than failure detectors operating in a small local area network. The wide area network problem is discussed in [10], and [13]. Each discussion presents protocol adjustments to address the problem of the failure detectors running across wide area networks. Failure detectors should be designed so that if they are not already efficient for wide area networks, they can be adjusted to operate in them with minimal increases in complexity and messages used for organization and coordination.

### **2.1.4 Computational Complexity**

Group applications that depend on a failure detection protocol may be running on a variety of devices. These devices can range from powerful servers and desktop PCs to PDAs, cell phones, or small sensor type devices. Smaller devices such as PDAs and sensors have limited computational power, memory, and battery life. Designing a failure detection component so that it is simple in terms of computations performed and memory needed will conserve the limited resources available on small devices. Since the failure detector will be minimizing its use of the CPU, which drains the battery supply, it will be conserving the battery supply as well. Applications running

on servers and PCs where resources may not be as much of an issue will still want to minimize CPU usage so that it is available for other applications.

Another point to consider is the fact that besides the messages sent by the failure detector, the application running on the hosts will be passing messages of some form between processes. If the failure detector is so complex that it demands a large amount of CPU processing time, less time will be available for processing the incoming message packets. In conclusion it is important that the design of a failure detection protocol takes these factors into consideration and conserves resources so they are available for other application demands.

## **2.2 Assumptions**

### **2.2.1 Network**

Network dynamics can drastically change the design of a failure detection protocol. If a protocol is designed for use over a reliable network layer, then it most likely will not be useful in an unreliable network setting. The protocols discussed and implemented in this thesis investigation were tested over a network of workstations in the RIT computer science department. The delay time of packets sent over this network was not varied in the experiments and was assumed to be minimal. The packet loss rate in the network was also assumed to be minimal and in most instances equal to zero. To introduce network unreliability into the testing environment, mechanisms for simulating network packet loss were added to the testing framework.

### **2.2.2 Protocol Messages**

All messages sent between nodes in the failure detection protocols is done so using UDP. Therefore packets are sent best effort but it is assumed that they reach their destinations in most cases. In the real world packets may be dropped at any moment. A protocol's ability to handle message loss will be tested by varying the network drop rate.

### **2.2.3 Point-to-Point Communication**

Throughout this thesis all nodes communicated by sending point to point messages. Broadcast and multicast capabilities would enhance the protocols discussed and improve their efficiency. The assumption is that such capabilities may not be available at all times and in all networks. So this thesis investigated protocol performance in the worst case scenario where only point-to-point communication was available.

#### 2.2.4 Failure Detection vs. Group Membership

A difficult task in failure detection is trying to determine whether a node is failed or is instead sleeping, processing other tasks, temporarily unavailable, etc. This topic is discussed in [14]. In this thesis the decision of when a node should be removed from the group is not relevant. The assumption is that this decision would be left up to an upper application layer. This layer would monitor the failure detector's suspect list and make group membership decisions based on how long a node appears in the suspect list, or how many times it has been added to the list. This thesis was only concerned with how a node is added to the suspect list.

#### 2.2.5 Group Sizes

The assumption for this thesis about group size was that any failure detection protocol should scale well to any group size, and therefore no target group size was used. The group size was not varied only when it was necessary to hold it constant to test other environment variables. The only limitation on group sizes was in the test environment where the number of hosts available was limited at times.

### 3 Available Protocols

#### 3.1 Heartbeat

A very basic failure detector algorithm is the Heartbeat protocol. The basic heartbeat algorithm is popular due to its simplicity as well as its speed in most cases. It is considered a *push model* algorithm since nodes try to send or *push* information to other nodes in the group. In the basic heartbeat algorithm nodes periodically send heartbeat messages to all other nodes. A heartbeat message is as simple as sending the identifier of the sending node. Detecting a failed node is just as simple. If a node stops receiving heartbeats from another node, it assumes that the silent node has failed. Heartbeat protocols are discussed in [5], [11], [3], [2], [8], [12], [4], [1], and [9].

A negative consequence of the simplicity is the amount of messages used in the protocol. Basic Heartbeat protocols do not scale well due to the large number of messages used. Numerous protocols exist building off of the basic heartbeat algorithm. An example of such a protocol addition is adding structure to the group, such as organizing the nodes into a tree. Other protocols try to use a centralized node or server for the heartbeat destinations, which reduces the number of messages used. The Heartbeat protocol discussed throughout this thesis refers only to the basic heartbeat algorithm.

### 3.1.1 Description

In a heartbeat scheme each member sends a heartbeat *"I am alive"* message to all other nodes at each  $T_{heartbeat}$  time interval. The time value used for  $T_{heartbeat}$  is a protocol parameter that may be predetermined or dynamically adjusted as the protocol operates. In this thesis investigation the time period was configured at node initialization.  $T_{heartbeat}$  values can range from milliseconds to seconds or even minutes. Each member maintains a list of all the other members in the group. The time stamp of the last received heartbeat is stored for each member in the list. An example of a group using the Heartbeat protocol is given in figure 1. Each node initializes its list of time stamps at time interval  $T_0$ . The figure shows node A sending heartbeat messages to all other nodes at time interval  $T_1$ . It is important to note that the figure focuses on the send procedure executed at node A only, but in reality all four nodes would be executing the same send procedure. An example of the code executed by node A is as follows.

#### Example Heartbeat Procedure at node A:

```
foreach  $n_j$  in node_list { /* node_list = { $n_0 = B, n_1 = C, n_{n-1} = D$ } */
    send message to  $n_j$ : "HEARTBEAT, A";
}
```

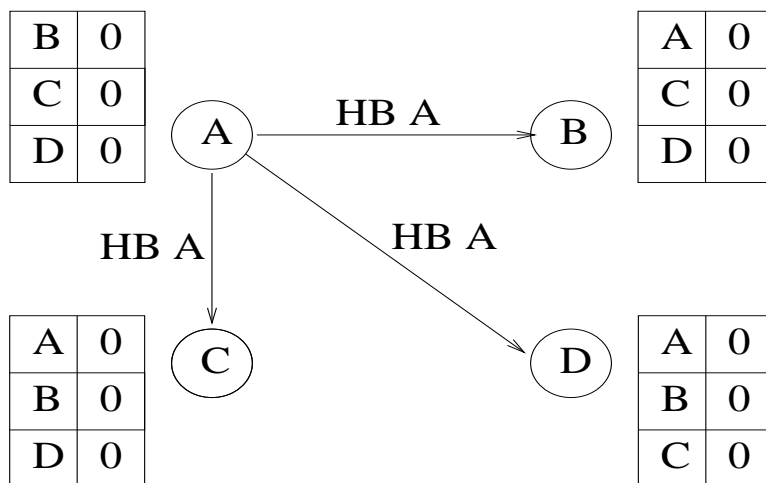


Figure 1: Node A Sending Heartbeat Messages

A heartbeat message is processed by examining the identifier of the sender. The receiver node then updates the corresponding time stamp in its list of time stamps. Figure 2 depicts the group situation after nodes B, C, and D process the heartbeat messages sent by node A. Since the nodes receive the messages at time interval  $T_1$ , they update the time stamp for A in their tables with time interval  $T_1$ . An example of the code executed by nodes B, C, and D is as follows.

```

Any time at node  $n_i$ :
if receive "HEARTBEAT,  $n_j$ " {
    update  $n_j\_timestamp$ ;
}

```

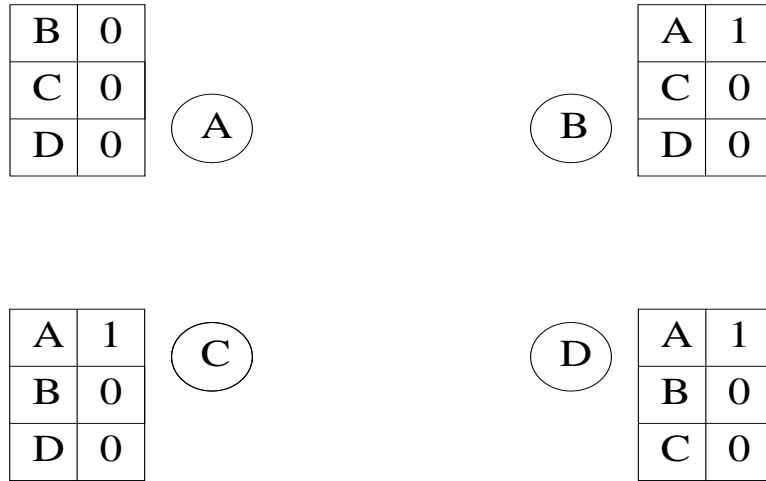


Figure 2: Nodes B, C, and D With Updated Lists

When a  $T_{timeout}$  interval of time has passed since the last heartbeat message has been received for a given node, that node is considered failed. The  $T_{timeout}$  time value is a protocol parameter similar to  $T_{heartbeat}$ . To check for a failure, each node scans its list subtracting each time stamp from the current time period. If the result is greater than  $T_{timeout}$ , that node is added to the suspect list. Figure 3 depicts a situation where the group detects a failure. In this example node C terminates just after sending heartbeat messages to the other nodes and processing heartbeats from B and D. Nodes A, B, and D have sent and processed messages sent and received at time interval  $T_6$ . The  $T_{timeout}$  value is equal to 3 (3 heartbeat intervals). Nodes A, B, and D check their lists and notice that the current time interval ( $T_6$ ) minus C's time stamp (2) results in a value (4) that is greater than the  $T_{timeout}$  period (3). As a result nodes A, B, and D add node C to their suspect list. An example of the code executed by A, B, and D is as follows.

**Example Timeout Check Procedure:**

```

foreach  $n_j$  in node_list {
    if ( current_time -  $n_j\_timestamp$  >  $T_{timeout}$  ) {
        add  $n_j$  to suspect_list;
    }
}

```

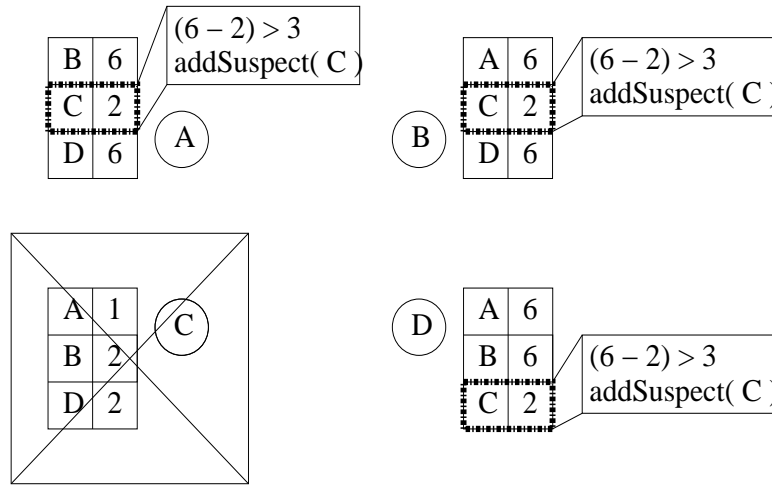


Figure 3: Nodes A, B, and D Detect C

Although A, B, and D have added node C to their suspect list, they continue to send heartbeat messages to C. Process C may be temporarily unavailable due to network problems, and may become available in the near future. Node C's host may be busy with other more urgent applications. It is assumed that the upper application layer will decide when to remove a node from the group, and will coordinate any removal procedure.

Whenever a node receives a message from another node it checks its suspect list and removes the node if it is there. Consider the situation where a node is temporarily unable to send heartbeat messages and the other nodes in the group have suspected it as failed. When the node recovers and continues sending heartbeat messages, the other nodes that receive the messages check their suspect lists and remove the previously suspected node.

### 3.1.2 Discussion

Since every node sends a heartbeat message to every other node every round,  $O(n^2)$  messages are sent every  $T_{heartbeat}$  period, where  $n$  is the number of nodes in the group. As the number of nodes in the group grows, the network will eventually be flooded with failure detection messages. Thus, the quadratic nature of the heartbeat algorithm limits its effectiveness as an efficient and highly scalable failure detection protocol.

To solve the message usage problem, Heartbeat protocols can use dedicated reliable nodes as heartbeat servers. Each member can send their heartbeats to the server node instead of all the nodes in the group. It is then the server node's responsibility

to track the heartbeat times of the group members and to detect when a node has failed. Although the centralized server based heartbeat schemes reduce the number of messages sent and the overall CPU usage on the node hosts, the servers are single points of failure and can render the failure detection system useless if they fail themselves. Using an election algorithm to assign new server nodes could help with this problem, but doing so may be expensive in terms of messages used and execution time.

Another alternative mentioned in the literature is to organize the group into a ring structure and send heartbeat messages around the ring. The problem with this structure is recovering from multiple node failures. The Heartbeat protocol discussed throughout this thesis refers only to the basic Heartbeat protocol. In summary the basic heartbeat algorithm is simple and fast, but it uses a large amount of network bandwidth and as a result is not highly scalable.

## 3.2 Ping

Another type of failure detection algorithm is the ping based strategy. Ping based failure detectors are similar to Heartbeat protocols in that they are also quite simple and easily implemented. Opposite of Heartbeat protocols and the *push model*, Ping protocols are examples of *pull model* algorithms in that they use messages that query nodes for information and require a response from the nodes. In a basic ping protocol each node periodically pings all other nodes. A ping message can be thought of as a request message. The receiver of a ping responds with an acknowledgment. If a node stops sending acknowledgments to ping messages, then the node is assumed to have failed. Ping protocols are discussed in [5], [12], and [9].

As mentioned with the Heartbeat protocols, improvements can be made to the basic algorithm to try and improve efficiency. Organizing the nodes into a some type of structure may improve efficiency, but doing so requires some means of coordination, as well as extra messages to maintain the structure as nodes join and leave the group. The Ping protocol discussed throughout this thesis refers only to the basic Ping protocol.

### 3.2.1 Description

Every round of the basic Ping protocol each node sends a ping "*Are you alive?*" message to every other node every  $T_{ping}$  time interval. The  $T_{ping}$  time interval is a protocol parameter similar to the  $T_{heartbeat}$  time interval defined in the Heartbeat protocol. Figure 4 depicts a group with node members A, B, C, and D. The nodes initialize their lists at time interval  $T_0$ . The figure shows node A sending ping messages to the other nodes in the group at time interval  $T_1$ . Nodes B, C, and D will also execute the same ping procedure as A at time interval  $T_1$ . An example of the code executed by node A is as follows.



**Example Ping Procedure at node A:**

```
foreach  $n_j$  in node_list { /* node_list = { $n_0 = B, n_1 = C, n_{n-1} = D$ } */  
    send message to  $n_j$ : "PING, A";  
}
```

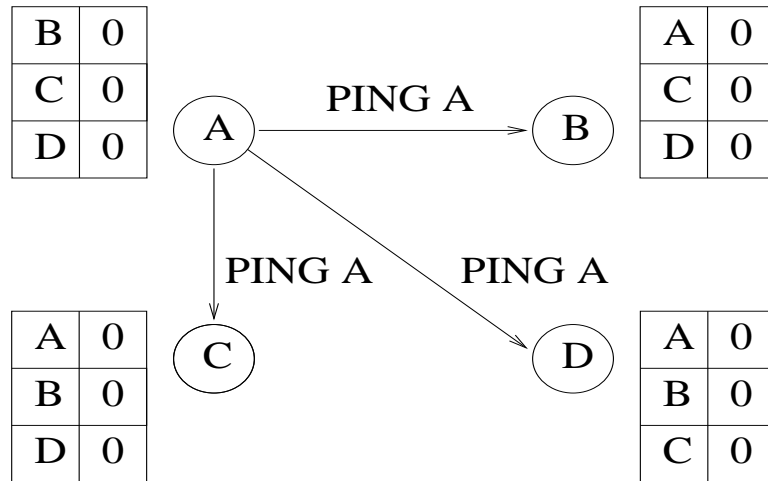


Figure 4: Node A Pings B, C, and D

When a member receives such a ping message it sends an acknowledgment "I am alive" message back to the sender. Figure 5 shows nodes B, C, and D responding to A's ping messages by sending acknowledgment messages to A. When a node receives an acknowledgment message, the pinged node is considered to be alive until the next ping period. In figure 5 node A records the acknowledgments received by B, C, and D by updating the list with a 1 for each received acknowledgment. An example of the code executed by nodes B, C, and D, as well as the acknowledgment process procedure executed by A are as follows.

**Any time at node  $n_i$ :**

```
if receive "PING,  $n_j$ " {  
    send message to  $n_j$ : "ACK,  $n_i$ ";  
}
```

**Any time at node  $n_i$ :**

```
if receive "ACK,  $n_j$ " {  
    update ACK for  $n_j$  in node_list;  
}
```

A node is added to a suspect list if an acknowledgment message has not been re-

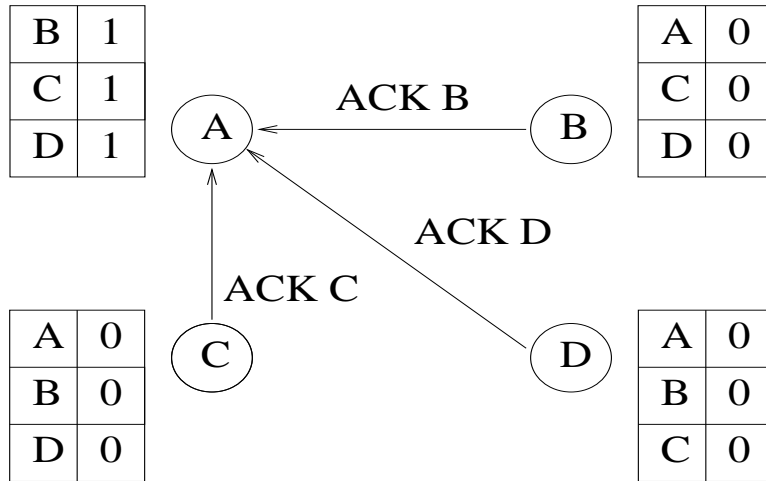


Figure 5: B, C, and D Send Acknowledgments to A

ceived within the  $T_{timeout}$  period since the time the ping was sent. Figure 6 shows nodes A, B, and D detecting the failure of node C. Nodes A, B, and D sent ping messages to C, and the  $T_{timeout}$  period has expired. Node C failed and did not respond with acknowledgment messages. Since the  $T_{timeout}$  period has expired, nodes A, B, and D scan their lists searching for missing acknowledgments. Nodes A, B, and D notice that they did not receive acknowledgments for their ping messages to C. A  $T_{timeout}$  period has passed since the ping was sent, so nodes A, B, and D add node C to their suspect lists. For the same reasons described in the Heartbeat protocol section, nodes A, B, and D continue sending ping messages to node C even though they have just added C to their suspect lists. The failure check procedure is as follows.

**Example Timeout Check Procedure at node  $n_i$ :**

```

/* Executed at  $T_{timeout}$  time units since last ping procedure. */
foreach  $n_j$  in node_list {
    if no ACK has been received from  $n_j$  {
        add  $n_j$  to suspect_list;
    }
}

```

The procedure for removing a node from the suspect is similar to the Heartbeat protocol. Whenever a node receives a message from another node it checks its suspect list and removes the node if it is in the list.

**3.2.2 Discussion**

An obvious difference between the Heartbeat protocol and Ping protocol is that the Ping protocol uses twice as many messages as the Heartbeat protocol. The Ping

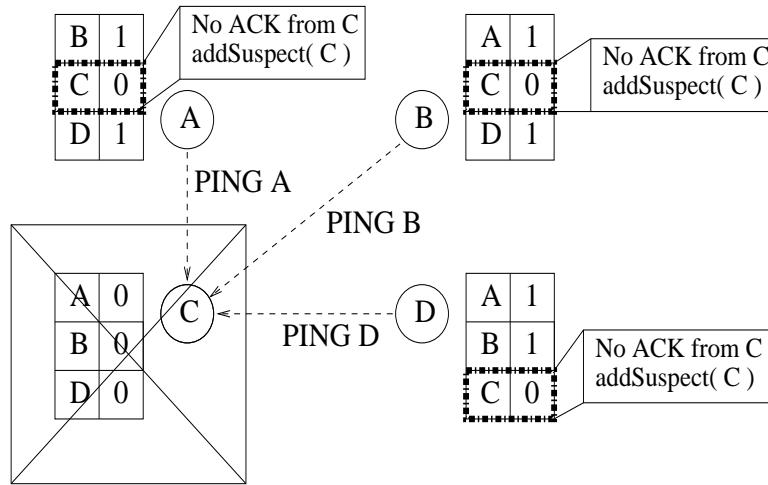


Figure 6: Nodes A, B, and D detect C

protocol still uses  $O(n^2)$  messages per round, but this estimation hides the factor of 2 that is actually present with the Ping protocol.

Similar to heartbeat schemes, ping protocols can use reliable dedicated nodes as pinging servers. These servers are responsible for pinging the other group members and determining whether or not a node has failed. Such a protocol design will reduce the number of messages used, but it also creates a few problems. Ping servers will create hot spots resulting in heavy network load in some areas. Also ping servers are single points of failure and may require coordination and more messages to be sent to select a new server node.

Similar to the Heartbeat protocol, the Ping protocol does not scale well to large group sizes. When the number of nodes in the group grows, the rate at which ping messages are sent must be adjusted so that the network is not overloaded with messages. As a result the speed of failure detections should decrease as group sizes grow. Although the Ping protocol can detect failures quickly and accurately, these properties suffer as the protocol scales for larger group sizes. The performance of the Heartbeat and Ping protocols will be compared to the performance of several other slightly more complex failure detection algorithms, such as the Randomized Ping protocol and the Gossip protocol.

### 3.3 Gossip

Gossip style protocols attempt to detect failed nodes by having each node *gossip* its view of the group to other random members. Gossiping means that all nodes periodically send their knowledge of the group to other random group members. The

knowledge that is sent is a list of information containing one line per node. In the basic Gossip protocol this line of information is nothing more than an identifier and integer counter pair. When a node compares its own list with a received list, it keeps the more recently updated lines. Before sending the list, each node updates the line that refers to itself. As long as each node keeps updating its own line in the list, the information will reach all members due to the gossiping nature of the group. When a node fails, it no longer is able to update its information. The other nodes detect the failed node when they realize that its line has become stale and has not been updated within a certain period of time.

The basic Gossip protocol is discussed in detail in [13]. The protocols presented in [10] as well as the ones in section 4 of [13], and section 4.3 of [12] build on the basic Gossip protocol and try to apply structured organization to the group to improve efficiency and reduce network load. Since this thesis investigation is focused on failure detection protocols not requiring extra processing or message passing for maintaining an organized structure, the Gossip protocol mentioned throughout this paper will refer to the basic Gossip protocol.

### 3.3.1 Description

The basis of the Gossip protocol is in the *gossip list* data structure maintained by every node. Each node in the group maintains a list of counters and time stamps for all other nodes in the group. Each node initializes all counters to zero, and all time stamps to the current time. An example group is depicted in figure 7. In this group the timers were initialized to time period zero. In each table the first column represents a node identifier, the second column is the counter for that identifier, and the final column is the time period.

At every  $T_{gossip}$  time interval, each member increments its own counter and randomly selects  $B$  other members to gossip its list to. The  $B$  value is an adjustable Gossip protocol parameter. A gossip message contains the individual's own counter along with the identifier and counter for every node in the list. As described in [13], the timing of the gossip procedure between nodes is not synchronized, but clock drift should be negligible. The gossip procedure for node A executing at time period 1 is depicted in figure 8. Assume that the group was configured with  $B = 2$ . In this case A randomly selected B and D to gossip to. An example of the code executed by node A is as follows.

#### Example Gossip Procedure at node A:

```
counter = counter + 1;
Randomly select B nodes from node_list;
foreach  $b_k$  in B {
```

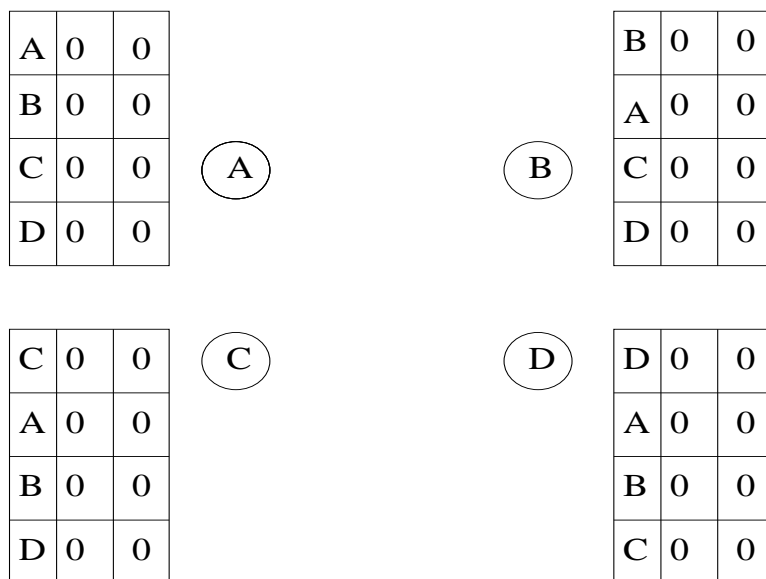


Figure 7: Example Gossip Group

```

send message to  $b_k$ :
    " $A_{identifier} = A_{counter}, n_{j\_identifier} = n_{j\_counter}$  (for all nodes  $n_j$  in node_list)"
}

```

When a node receives a gossip message it compares the list in the message to its own list. If a counter value for a given identifier in the message is larger than the corresponding counter value in its own list, the new value is kept and the time stamp is updated accordingly. Counters will eventually overflow, so protocol implementations should be designed so that they adjust to reset counter values. Figure 9 shows the group situation after B and D processed A's message and updated their lists. Nodes B and D would compare the message from A with their own lists and notice that the counter for A in the message is greater than the value in their own tables. They would then store the higher value in their table and update the time stamp. In this case B and D use time period 1 for the time stamp update. An example of code executed by nodes B and D is as follows.

#### Example Packet Process Procedure:

```

foreach  $n_{j\_identifier} = n_{j\_counter}$  pair in packet  $P$  {
    Obtain  $n_k$  in node_list where  $n_{j\_identifier} = n_{k\_identifier}$ ;
    if (  $n_{j\_counter} > n_{k\_counter}$  ) {
         $n_{k\_counter} = n_{j\_counter}$ ;
        update  $n_{k\_timestamp}$ ;
    }
}
}

```

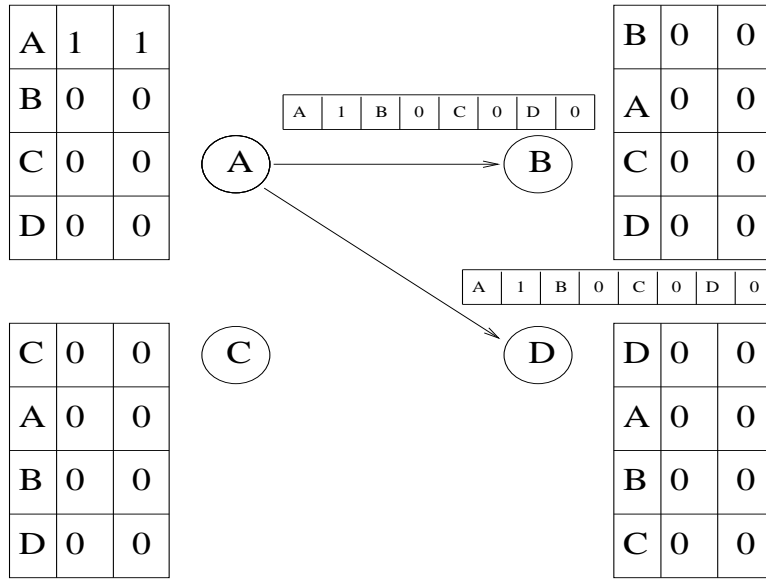


Figure 8: Node A Gossip Procedure

To detect faulty nodes, every node periodically scans its list searching for stale counters. A node is added to the suspect list if its counter has not been updated within the  $T_{timeout}$  interval. Figure 10 depicts a situation where the nodes in the group detect a failure. The group situation is as follows. The current time period is 8. Node C failed in time period 3, just after updating its own counter. Nodes A, B, and D have updated their counters and sent gossip messages that are currently on their way to their destinations. The  $T_{timeout}$  value is equal to 4. Nodes A, B, and D scan their lists and notice that the time stamp for node C is from time period 3. Each node calculates that C's time stamp of 3 subtracted from the current time period (8) results in a value (5), which is greater than the  $T_{timeout}$  period of 4. Nodes A, B, and D all add node C to their suspect lists. An example of code executed by nodes A, B, and D is as follows.

**Example Timeout Check Procedure:**

```

foreach  $n_j\_identifier = n_j\_counter$  pair in node_list {
  if (  $current\_time - n_j\_timestamp > T_{timeout}$  ) {
    add node  $n_j$  to suspect_list;
  }
}

```

Although C has been added to the suspect lists, nodes A, B, and D do not stop sending messages to C. Node C may be currently unavailable due to a network failure that may be fixed in the next few time periods, or it just may be busy with other

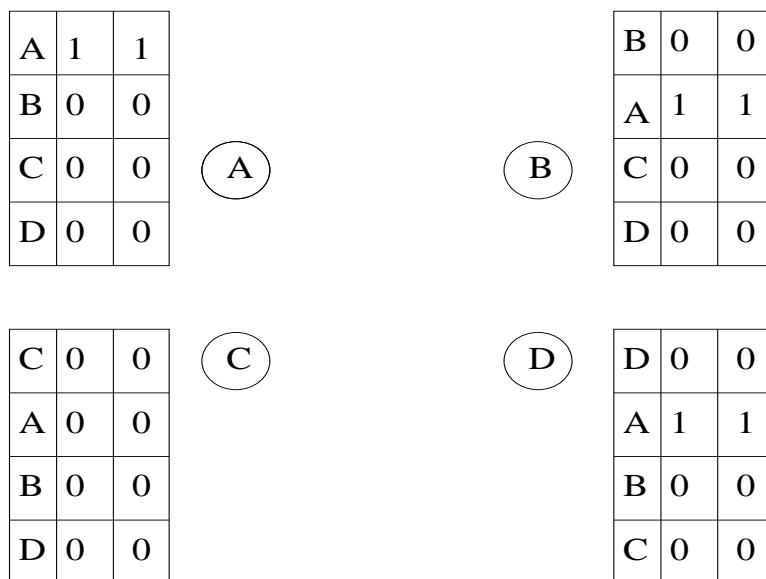


Figure 9: B and D With Updated Lists

processes. The assumption is that an upper layer of the application will decide when to remove a node from the group, and will coordinate the effort to take care of the removal. The procedure for removing a node from a suspect list is simple. Whenever a node receives a Gossip message from another node it checks its suspect list and removes the node from the list if it is there.

### 3.3.2 Discussion

As each node "infects" other nodes with information about the group, a unified view of the group is maintained allowing a low average detection time of failures. Adjusting the  $B$  parameter changes the rate at which information is passed around in the group. A higher  $B$  value means more nodes are "infected" each  $T_{gossip}$  period.

Each  $T_{gossip}$  period  $B \cdot n$  messages are sent across the network. The minimal amount of messages occurs when  $B$  is equal to 1. In the case exactly  $n$  messages are sent each  $T_{gossip}$  period which is  $O(n)$ . The maximum amount of messages occurs when  $B$  is equal to  $n-1$ . In this case  $n^2 - n$  messages are sent each  $T_{gossip}$  period which is  $O(n^2)$ . Setting  $B$  to the smallest value possible without sacrificing accuracy or detection times minimizes network load.

One might be quick to conclude that by using the optimal  $B$  value, the Gossip protocol can produce accurate detection times similar to the Heartbeat and Ping protocols using only  $O(n)$  messages compared to  $O(n^2)$ . The detail that is overlooked in this analysis is that every gossip message consists of all  $n$  (identifier, counter) pairs. As

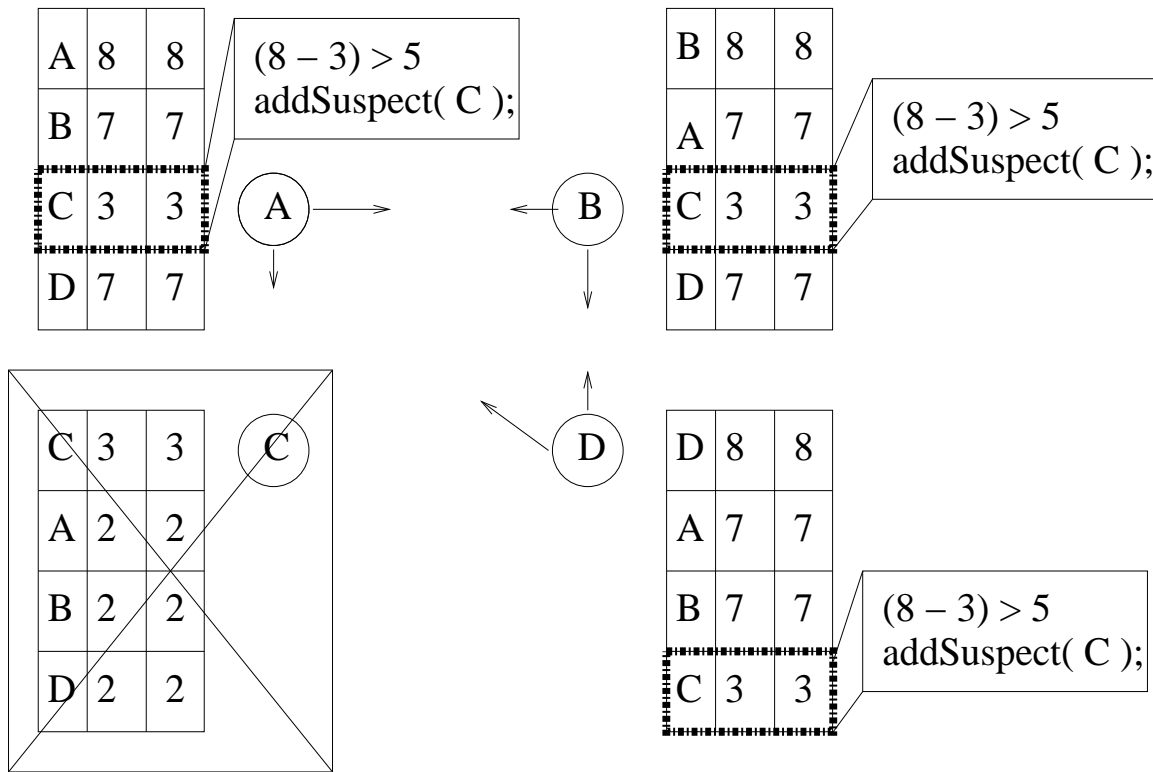


Figure 10: Nodes A, B, and D Detect C

n grows with the size of the group, the size of the messages sent grows proportionally as well. If the size of the messages is greater than the maximum payload of the underlying transport layer packet, then multiple packets will be needed to transport the gossip message. This can introduce more network stress than expected.

As discussed in [13] and [10], more efficient Gossip protocols can be constructed by making adjustments to the basic protocol. Instead of randomly selecting the nodes to gossip to each round, the nodes can be selected in a more deterministic fashion taking into account the network topology. This can reduce network load across heavy traffic areas without sacrificing protocol accuracy. Although not implemented in this thesis investigation, it is important to mention these extended Gossip protocols and that they have been presented and analyzed in the available research.

### 3.4 Protocol Proposed by Chandra, Goldszmidt, and Gupta

An interesting failure detection protocol has been proposed by Chandra, Goldszmidt, and Gupta. Their protocol is unique in that it uses a randomized ping strategy to obtain the completeness and efficiency properties that are required in a failure detection protocol. The protocol uses a lower number of messages compared to the basic



Heartbeat and Ping protocols, which should result in a more scalable protocol.

Instead of pinging every node in every round, each node only selects one node to ping. If no acknowledgment is received, the node executes a "swarm" procedure. The swarm procedure randomly selects a few other nodes in the group and sends them a message asking them to ping the node that timed out. The nodes ping the suspicious node for the original node. If an acknowledgment is received, they forward it to the original node. If the original node does not receive an acknowledgment from any of the swarm members, it adds the faulty node to its suspect list. The swarm procedure double checks a possible failure, increasing the accuracy of the protocol. The combination of reduced messages and increased accuracy should result in an efficient failure detector. Chandra, Goldszmidt, and Gupta's protocol is discussed in depth in [9].

### 3.4.1 Description

The basis of the algorithm is the random ping procedure. Every  $T_{ping}$  time interval each member randomly selects another member in the group to ping. If a node ever receives a regular ping, the node responds by sending an acknowledgment to the sender. Figure 11 depicts a group using the Random Ping protocol. The left side of the figure (a) shows the nodes pinging other randomly selected members. In this situation each message reaches its destination. The right side of the figure (b) shows the nodes responding to the pings by sending acknowledgment messages. Sample code executed by the nodes is as follows.

#### Example Random Ping Procedure at node $n_i$ :

```
Randomly select a node  $n_j$  from node_list;  
send message to  $n_j$ : "PING,  $n_i$ ,  $n_j$ ";
```

#### Any time at node $n_i$ :

```
if receive "PING,  $n_m$ ,  $n_i$ " {  
    send message to  $n_m$ : "ACK,  $n_m$ ,  $n_i$ ";  
}
```

If the sending node does not receive an acknowledgment message within the  $T_{timeout}$  interval, it randomly selects  $k$  other members to ping the node under suspicion. The  $k$  value is an adjustable protocol parameter. Figure 12 depicts a situation where node A executes the swarm procedure. Node A sent a ping message to node C and did not receive an acknowledgment. In this example the nodes were configured with  $k = 2$ . Node A selects nodes B and D for the  $k$  members. Since B and D are the only nodes available, they are selected for the swarm procedure. In larger groups the  $k$  nodes would be selected randomly. In the figure, node A sends ping request messages to B

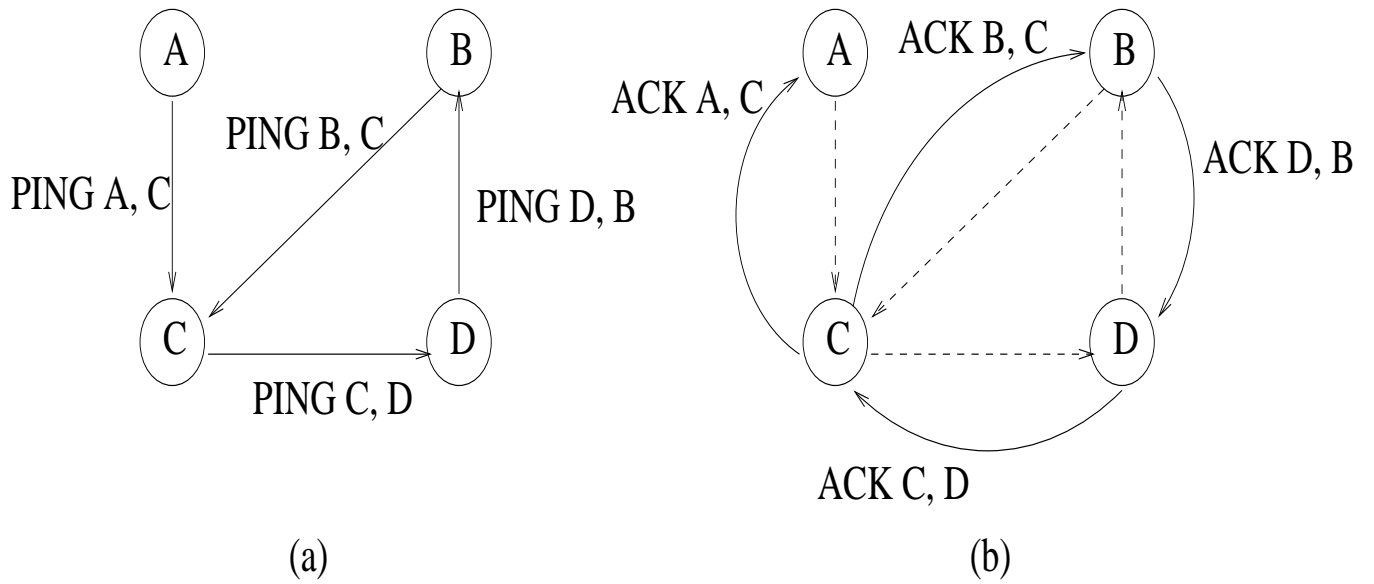


Figure 11: Sample Round of Random Ping Protocol

and D. An example of the code executed by node A is as follows.

**Example Ping Request Procedure at node A:**

```

if no "ACK,  $n_i$ ,  $n_j$ " message has been received within  $T_{timeout}$  period {
  Randomly select  $k$  nodes from node_list;
  foreach  $n_k$  in set of  $k$  nodes {
    send message to  $n_k$ : "PING_REQ,  $n_i$ ,  $n_j$ ";
  }
}

```

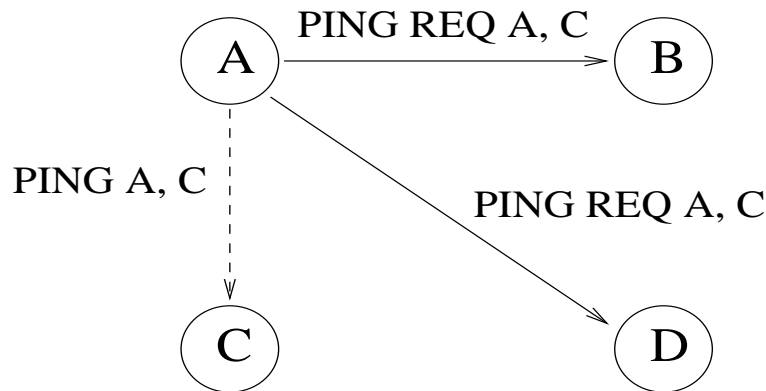


Figure 12: Example Swarm Procedure

If a node ever receives a ping request message, it pings the suspicious node for the requester. Figure 13 shows nodes B and D sending ping messages to node C in response to the request messages received from node A. Besides including their own identifiers in the ping message, they also include the identifier (A) of the node that executed the swarm procedure at the end of the message. A sample of the code executed by nodes B and D is as follows.

```

Any time at node  $n_i$ :
if receive "PING_REQ,  $n_m$ ,  $n_j$ " {
    send message to  $n_j$ : "PING,  $n_i$ ,  $n_j$ ,  $n_m$ ";
}

```

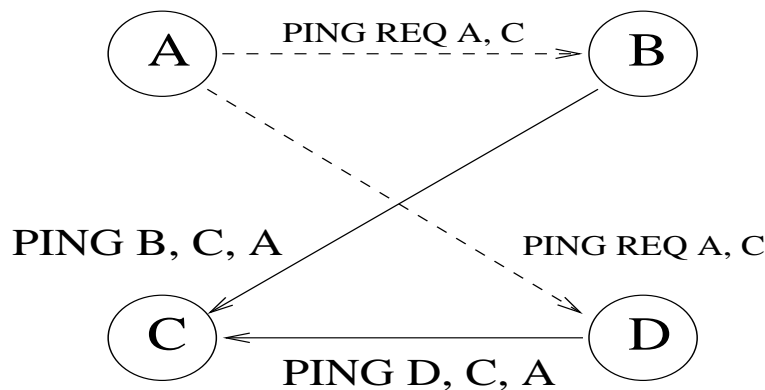


Figure 13: Nodes B and D Sending Pings to C

If the suspect node receives a ping message from one of the  $k$  members, it replies with an acknowledgment. If any of the  $k$  members receives an acknowledgment, it will forward the acknowledgment back to the original sender. Figure 14 depicts these procedures. Consider a network with a high drop rate. The original ping packet from A to C never reached its destination. This caused A to execute the swarm procedure as seen in figure 13. The left side (a) of figure 14 shows node C responding to the ping sent by node B. The ping from D to C was lost in the network and never reached C. The right side (b) shows node B forwarding to A the acknowledgment it received from C. In the normal ping protocol node A would have incorrectly suspected node C as failed. In Chandra, Goldszmidt, and Gupta's protocol, the swarm procedure prevented an incorrect detection. This example helps illustrate the accuracy of their protocol. Sample code executed by the nodes in figure 14 is as follows.

```

Any time at node  $n_i$ :
if receive "PING,  $n_m$ ,  $n_i$ ,  $n_j$ " {
    send message to  $n_m$ : "ACK,  $n_m$ ,  $n_i$ ,  $n_j$ ";
}

```

}

**Any time at node  $n_i$ :**

```

if receive "ACK,  $n_i$ ,  $n_j$ ,  $n_m$ " {
    send message to  $n_m$ : "ACK,  $n_m$ ,  $n_j$ ";
}

```

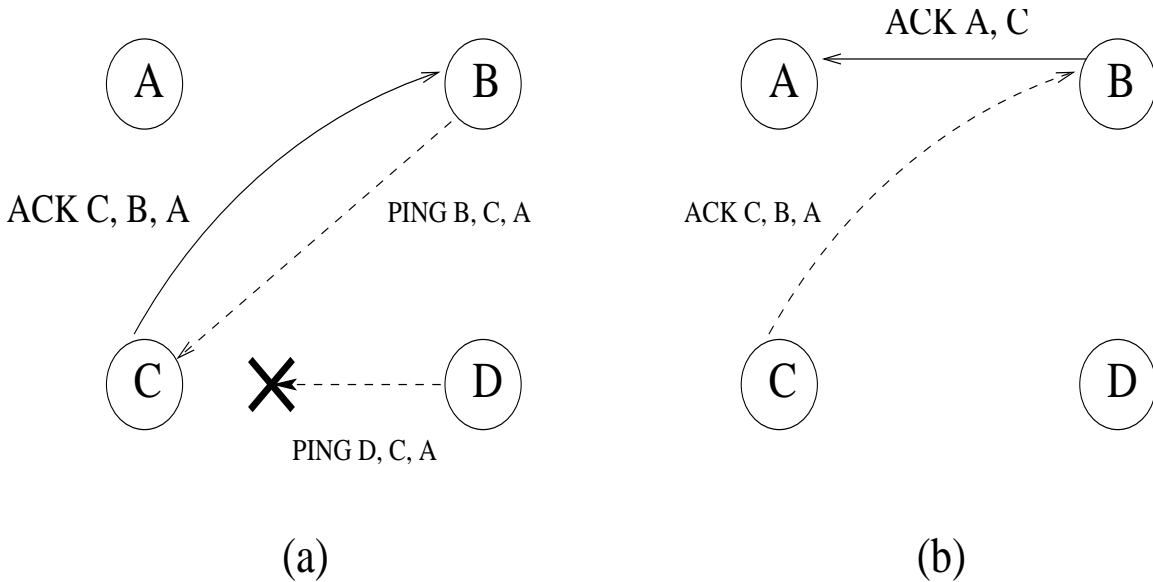


Figure 14: (a) C Responding to Swarm Ping, (b) B Forwarding Response to A

If the original sender does not receive an acknowledgment from any of the  $k$  subgroup members within the  $T_{timeout}^i$  interval, it adds the faulty node to the list of suspected nodes. Consider the situation where the network is reliable and node C has failed. Node C never responded to A's ping message so A initiated the swarm procedure seen in figure 13. Since C failed, nodes B and D never receive any acknowledgments to forward back to node A. As a result node A never receives an acknowledgment within the timeout period and adds C to its suspect list. Figure 15 depicts this situation. An example of the code executed by node A is as follows.

**Example Final Timeout Procedure at node  $n_i$ :**

```

if no "ACK,  $n_i$ ,  $n_j$ " message has been received within  $T_{timeout}^i$  period {
    add node  $n_j$  to suspect_list;
}

```

To keep the suspect list as up to date as possible every node checks its suspect list when it receives a message. If the node that sent the message is in the receiver's suspect list the receiver removes it. This is how a node is removed from another

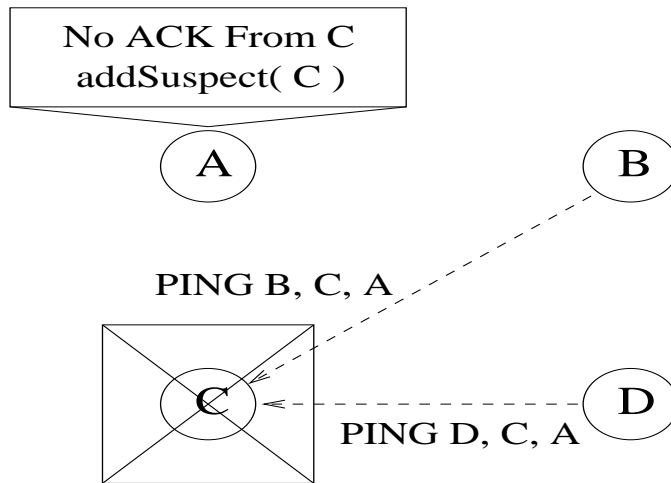


Figure 15: Node A Detects C

node's suspect list.

### 3.4.2 Discussion

In a perfect scenario where all messages are delivered and no processes terminate unexpectedly, Chandra, Goldszmidt, and Gupta's protocol will use one message for a ping and one message for an acknowledgment per node for each ping round. This results in  $O(n)$  messages in an ideal setting. In a setting with an unreliable network and processes that may fail, as more packets are lost and processes fail, the number of messages should grow due to the use of the auxiliary  $k$  ping request procedure.

In their paper Chandra, Goldszmidt, and Gupta present proofs for the expected detection time of a node failure, the probability of a detection error, as well as proofs that the protocol meets the completeness, speed, and accuracy requirements of an efficient failure detector. These proofs will not be discussed here but it should be noted that the detection time discussion is based on the average time needed for a node to detect a failure. A point that is not discussed is the maximum time that may be needed for a node to detect a failure.

There are several outcomes that may arise due to the random selection of nodes to be pinged each round. A node may get lucky and randomly select a node that has just failed. In this case the node will quickly detect the failure. Another case may be that a node is quite unlucky and does not select the failed node for a long period of time. What this means is that the protocol has the unfortunate property of having a high maximum detection time. Chandra, Goldszmidt, and Gupta touch on this problem when they mention possible ways to alter the protocol so that it is deterministic instead of random when choosing nodes. This would result in a known

upper bound on detection time.

Due to the low message usage of Chandra, Goldszmidt, and Gupta’s protocol, scaling for large group sizes should not be a problem. The only apparent drawbacks of the algorithm are the maximum detection time issue, and as mentioned in [9] the fact that if the group spans across different network regions, the routers and gateways will need to handle the protocol message load. Adding some kind of group structure to the protocol is a possible solution to this problem, as mentioned with the previously discussed protocols, but will not be investigated in this thesis.

## 4 Aggressive Protocol

The main focus of this thesis was the development of a new failure detection protocol. This protocol would be based on modifications of the basic design of Chandra, Goldszmidt, and Gupta’s random ping strategy. The hypothesis was that the modifications would eliminate the maximum detection time problem and reduce overall average detection time while minimizing any increases in messages sent and CPU usage.

### 4.1 Basis

In every round of Chandra, Goldszmidt, and Gupta’s protocol, each group member randomly selects another member to ping. A consequence of this is that some members of the group may be pinged more often than others. This results in some nodes having to handle a larger portion of the computational burden. Another consequence is that time and messages may be wasted pinging nodes in the group that are more reliable than other nodes that are more susceptible to failure. Since the selection of the  $k$  nodes to ping a suspected node is also done in a random manner, less reliable nodes may be selected instead of more reliable ones, resulting in a larger number of failure detection errors.

Another negative effect of the random ping selection process is that a node may not ping a given node for a very long period of time. If that node had failed, then it would take a long time to detect the failure. This results in the high maximum and average detection times of Chandra, Goldszmidt, and Gupta’s protocol.

The proposed new protocol would have several changes that would eliminate these problems.

## 4.2 Modifications

The goal of the protocol modifications was to eliminate inefficiencies that arise due to the randomness of the protocol, as well as alter the ping selection process so that it is deterministic and targeted for nodes more likely to fail. While developing these modifications it became apparent that this approach appears to be flawed. This issue will be examined in section 4.3. Sections 4.2.1 through 4.2.5 present each of the proposed modifications with discussions of their merits or downfalls.

### 4.2.1 Round-Robin Selection

The first improvement was to replace the random ping selection procedure with a round-robin type procedure. This would ensure that every member receives a fair amount of ping messages without overloading any given member of the group. An example data structure for the group view list could be a simple array. Each member starts with the member index immediately following its own index. At each round the array index is incremented. When the end of the array is reached, the index is reset to 0 and starts from the beginning of the array.

#### Example Ping Selection Procedure at node $n_i$ :

```
node_to_ping = group_list[ next_index ];  
next_index = ( next_index + 1 ) mod ( group_size - 1 );  
sendPing( node_to_ping );
```

Unlike the random ping selection procedure, the worst case detection time is known. For a group of size  $n$  the worst case detection time would be equal to the number of nodes multiplied by the ping interval time plus the timeout period for the last node resulting in  $((n - 1) * T_{ping}) + T_{timeout}$ . Using this selection procedure the worst case detection time will occur when a node fails just after it sends a node an acknowledgment. The node that received the acknowledgment will now have to work through it's list of nodes before it pings the faulty node. As the number of nodes in the group grows it will take a node longer to cycle through and ping each of the nodes in its node list. For very large groups this can be as much of a problem as randomly pinging nodes. The proposed burst notice improvement presented in section 4.2.4 is an attempt to eliminate the maximum detection time problem in large groups.

### 4.2.2 Node Characterization

Another proposed improvement required each node to maintain a reliability rating for the other nodes in the group list. This rating was based on several factors such as the percentage of ping responses, ping response time, and the number of times the node had failed to respond to initial ping messages. Each node also stored the best value for each category. To calculate the factor for a node, each of its values are compared

to the best values for the corresponding category. The percentages are then totaled to produce the node's rating. This rating could be used in the ping selection procedure which is discussed in the next section.

The concept of node characterization can be expanded by adjusting the rating for the host device that each node runs on. If the device is small with limited resources, then the rating for that node could be adjusted accordingly to reduce the number of times it is pinged. This could reduce resource consumption on limited host devices. This concept has not been implemented or tested, but functionality exists in the test framework to define host devices.

### 4.2.3 Selection Based on Characterization

In the round-robin ping selection procedure, each node considered the reliability rating of the current ping index. The probability that any given node will be selected increases as that node's reliability rating decreases. The idea behind this is that less reliable nodes should be pinged more often than reliable ones. As a result detection times should decrease since the less reliable nodes are more likely to fail.

One overlooked detail in the thought process is the situation in the group environment that will take place most of the time. This is the case when all devices and the network behave in a normal and predictable way. In this situation all nodes will have a relatively equal rating, and as a result the ping selection procedure skips through the nodes more often than actually selecting a node to ping. This wastes CPU time and actually increases detection times. After these effects were observed while working on the improvement, the ping selection procedure was changed back to the original simple round-robin selection described earlier.

The reliability rating calculation could still be useful in other areas of the algorithm. Parameters were added to the algorithm to track the top  $X$  nodes and bottom  $Y$  nodes in terms of their reliability ratings. The idea behind this was that the  $k$  nodes used in the swarm procedure could quickly be selected from the pool of top nodes, and the rest randomly selected if  $k > X$ . Separate timer tasks were used to ping the bottom  $Y$  nodes at an interval determined by a protocol parameter. The effects of these two protocol additions are discussed in section 4.3.

### 4.2.4 Burst Notices

To detect a node failure, a node must first ping the faulty node, timeout and send ping requests to  $k$  nodes, and then wait for an acknowledgment. If no acknowledgment is ever received then the original sending node adds the faulty node to its suspect list. An adjustment was made to the algorithm so that when a node adds another node to its suspect list it also "bursts" the information to all other nodes. The nodes that



receive such a burst message first check their own suspect list. If it already contains the suspect node, then they do nothing. Otherwise they ping the suspicious node to confirm the detection. This reduces the amount of work needed to detect the failure on their own, as well as the time needed to work their way through their group list using the round-robin ping selection.

The burst notice addition was implemented and tested. In a perfect network environment where the network drop percentage was equal to zero, the burst notice dramatically improved detection times, almost to the point where maximum, minimum, and average detection times were equal. The addition also made minimal increases in terms of protocol messages used. Such an improvement seemed too good to be true, but as with other improvement ideas there was a catch.

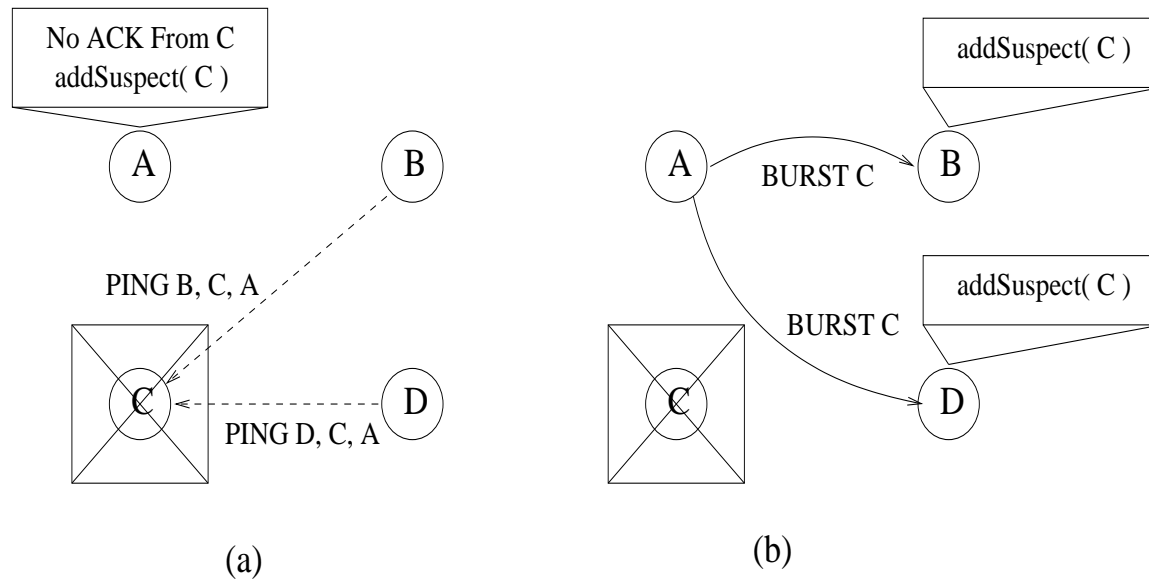


Figure 16: (a) B and D ping node C, (b) C sending ACKs to B, D, and A

The burst addition caused problems when used in a network where packet loss can occur. As packet loss increases, more incorrect detections take place, resulting in numerous burst notices. Each burst notice uses  $n-2$  messages to notify the rest of the group (excluding the sending node and the faulty node), as well as  $n-2$  more messages for each of the nodes receiving the burst notice to send a confirming ping to the faulty node. The impact the procedure has on the overall protocol performance increases as the network drop rate increases. The increase in messages used actually hurts performance as the network becomes more faulty. As a result of this observation, the burst procedure was changed so that when a node receives a burst notice it immediately adds the suspicious node to its suspect list if it is not already there. Part (a) of figure 16 shows node A adding node C to its suspect list after failing to receive an

acknowledgment from C during a swarm procedure. Part (b) shows node A sending burst notices to nodes B and D. When nodes B and D receive the burst notices from A, they add C to their suspect lists (assuming the suspect lists do not already contain C). This eliminates the extra ping message used to confirm the detection. The consequence of removing the confirming ping is that if a node mistakenly detects a correct node as failed and notifies the rest of the group, then the rest of the group will also have the incorrect detection in its suspect list. The basis of the Random Ping protocol is its improved efficiency using the swarm procedure. The burst notice modification is only an improvement if the accuracy of the underlying Random Ping procedure holds.

### 4.3 Discussion

While working on the protocol modifications it became apparent that the idea behind most of them was flawed. Maintaining information about the other nodes for calculating a reliability rating may seem like a good idea, but the way it was being used was not helping the performance of the protocol. Making the protocol aggressive is not the problem. The problem is the direction in which the aggressiveness is being applied. The original intent was to make the protocol aggressive in that it would attempt to ping weaker nodes more often than the reliable nodes, resulting in faster detection times. A consequence of this aggression is that an already weak system might crash as a result of the ping overload. A failure detection algorithm should accurately detect node failures as fast as possible, but it should not be a burden on weak systems or be the root cause of a crash.

After this realization, the original idea behind the modifications was abandoned and some time was taken to reflect on its flaws. Time was also taken to review the other protocols. The Heartbeat and Ping protocols are aggressive in that they send messages to every node every round. This makes them fast, but they use a large number of messages as a consequence. The Gossip protocol is accurate and fast for a different reason. Its performance properties arise from the information the nodes receive from each other about the group, instead of trying to gather it all themselves. This knowledge sharing property is the beauty of the Gossip algorithm. The price the Gossip algorithm pays for such knowledge sharing is the size of the messages sent between the nodes. The Random Ping protocol's swarm procedure provides accuracy, but it lacks speed since only one node is pinged per round.

An ideal improvement for the Random Ping protocol would be the addition of knowledge sharing in some way. The burst procedure is an attempt at this. In the final form of the burst procedure if a node completes the swarm procedure it sends a burst notice to the other  $n-2$  nodes. When a node receives a burst procedure it adds the faulty node to its suspect list if it is not already there. This knowledge sharing should

improve the average detection time of the Aggressive protocol. As mentioned previously, this burst notice procedure relies on the accuracy of the underlying swarm detection procedure. Functionality was added to the Aggressive protocol to improve the accuracy of the swarm procedure.

The swarm procedure was modified as follows. Whenever a node receives a ping from one of the  $k$  members of a swarm procedure it sends an acknowledgment to the node that pinged it, but it also sends an acknowledgment to the node that initiated the swarm procedure. Part (a) of figure 17 shows a group situation where node A has initiated a swarm procedure on node C. Node A sent ping request messages to nodes B and D. Nodes B and D respond by pinging node C. Part (b) of figure 17 shows node C sending acknowledgment messages to nodes B and D. It also shows node C sending an acknowledgment back to node A for each of the pings received from B and D. This will improve the accuracy of the protocol in a network with high packet

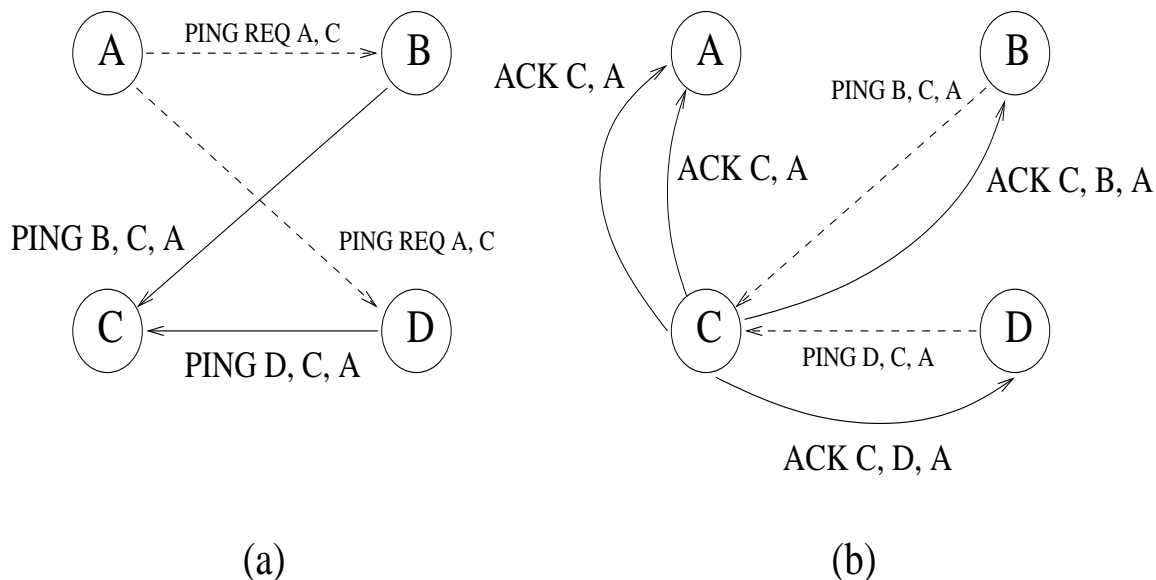


Figure 17: (a) B and D pinging node C, (b) C sending ACKs to B, D, and A

loss. The node receiving a ping from one of the  $k$  nodes needs to send the acknowledgment back to that node, and when it is received it is forwarded back to the node that started the swarm procedure. In the group example seen in figure 17 B and D will have to forward the acknowledgments received from C back to A. By sending an acknowledgment directly back to the swarm procedure initiator, as well as to the  $k$  members, the likelihood that the initiator receives an acknowledgment increases. The idea behind this procedure is that the increased accuracy will outweigh any negative impacts caused by the increase in messages sent.

It its final state the Aggressive protocol has the following features; the round-robin

ping selection procedure, burst notification with immediate suspect list addition, and the modified swarm procedure with acknowledgments sent to the  $k$  members as well as the swarm procedure initiator. This is the Aggressive protocol that was used in the tests presented in sections 8 and 9.

## 5 Protocol Comparisons

Consider the designers of the applications discussed in the introduction. Assume they both understand why they need a failure detection component in their application. They have examined the protocols discussed in the previous sections. All that is left for them to do is to select the best protocol and implement it in a failure detector component for their application. This sounds like a fairly simple process. Just choose, code, and integrate the finished component with the rest of the application. A question one may have is how did they decide which protocol is best?

Having a metric for comparing the protocols would be ideal for deciding which protocol is the best. The metric would produce some value allowing easy comparison. Part of this thesis was to develop such a metric. The metric would be based on messages used, accuracy (minimal number of errors), speed of detection, and computational complexity. After examining the protocols one can see that each of them are unique in their own way. They each have strengths and weaknesses. Having a single metric for comparison may hide characteristics that might reveal themselves if used under certain circumstances. Such a situation could be devastating for an application. Instead of comparing the protocols with a single metric, it may be better to compare them using one factor at a time. Performance in these areas also determines a protocol's ability to scale to large group sizes as well as the amount of stress the protocol will have on the network. Designers may find it best to select a protocol based on what factors they feel are most important for the performance of the application. They will also need to consider the environment the application will be running in. The results from the tests presented in section 8 would be very useful in determining what protocol is best in a given environment.

### 5.1 Messages Used

An important characteristic for comparing failure detection protocols is the number of messages each protocol uses. The testing environment used in this thesis did not have the capability to vary the amount of network delay. It did have the capability to vary the percentage of packets dropped in the network. As a result, only the amount of packets dropped in the network will be used when comparing the protocols.

The Heartbeat and Gossip protocols send the same number of messages per round, no matter how many packets are lost in the network. The Heartbeat protocol sends

$n^2-2$  messages per round, and the Gossip protocol sends  $Bn$  messages per round. The Ping protocol's total messages sent depends on the percentage of packets that reach their destination. Acknowledgment messages are only sent by the nodes that actually receive ping messages. If  $p$  is the percentage of packets that reaches their destination, then the Ping protocol will send  $(n^2-n)(1+p)$  messages per round (pings plus acknowledgments).

Calculating the effect of the network drop percentage on the Random Ping and Aggressive protocols is more complex than the Heartbeat, Ping, and Gossip protocols. Since the Aggressive protocol is an extension of the Random Ping protocol, it makes sense to evaluate the Random Ping protocol first. The Random Ping protocol starts out the same as the Ping protocol. At the beginning of the round every node randomly selects another node to ping. If  $p$  messages reach their destination then the total number of acknowledgments sent is equal to  $pn$ . A swarm procedure will be initiated for the each of the nodes where no acknowledgment was received. This means  $kn(1-p)$  ping request messages are sent. Since  $p$  times the number of ping request messages reach their destinations,  $pkn(1-p)$  total swarm ping messages are sent. Continuing with the same logic, there will be  $p^2kn(1-p)$  swarm acknowledgments sent, and  $p^3kn(1-p)$  swarm acknowledgments forwarded back to the swarm procedure initiators. In total there are  $n(1 + p + k - p^4k)$  messages sent per round of the Random Ping protocol.

The swarm procedure modification results in twice the number of swarm acknowledgments sent, resulting in a new total of  $n(1 + p + k + p^2k - p^3k - p^4k)$  messages sent per round. The next step is to calculate the messages sent as a result of the burst procedure modification. The total number of swarm acknowledgments received is equal to  $p^3k - p^5k$ . A burst notice will be sent for every node that failed a swarm procedure and was added to a suspect list, resulting in  $(n - 2)(1 - p - p^3k + p^5k)$  total burst messages. Since it is impossible to send negative messages, the burst messages are only included in the total when the quantity  $(1 - p - p^3k + p^5k)$  is positive. So the total number of messages sent by the Aggressive protocol per round is  $n(1 + p + k + p^2k - p^3k - p^4k + (n - 2)((1 - p - p^3k + p^5k) > 0) ? (1 - p - p^3k + p^5k) : 0)$ .

An assumption made in this evaluation is that the probability that any given packet is dropped is completely uniform. What this means is that for every set of  $k$  messages sent in a given swarm procedure, exactly  $pk$  messages reach their destination. Figure 18 compares the total number of messages sent by each protocol as the reliability of the network decreases. For this example the number of nodes in the group is 100,  $B = 3$  for the Gossip protocol, and  $k = 3$  for the Random Ping and Aggressive protocols. This graph, as well as the graph presented in figure 19, were generated using the protocol functions presented in this section. The graph shows how the burst procedure can have a negative impact when the reliability of the network decreases beyond a

certain threshold. This threshold depends on the value of  $k$ . In this example for  $k = 3$ , the burst procedure does not start to increase the total number of messages sent until the message delivery percentage drops below 55%. To configure the Aggressive protocol for optimal performance, a value of  $k$  would need to be selected so that the burst procedure does not have a negative impact based on the given network conditions. In this example  $k$  would need to be reconfigured once the percentage of messages delivered drops below 55%.

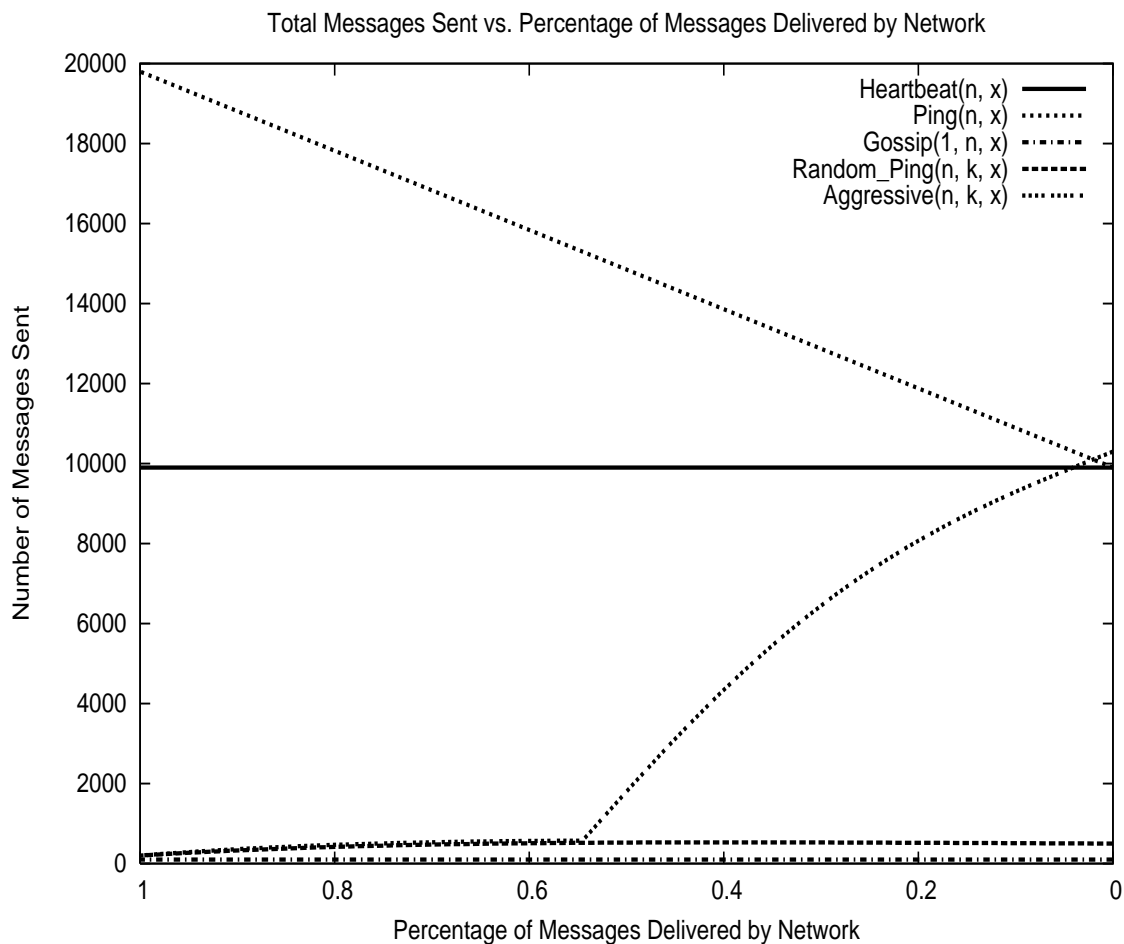


Figure 18: Total Messages Sent by the Protocols

Figure 19 shows how the total messages sent grows as the value of the  $B$  (Gossip) and  $k$  (Random Ping and Aggressive) parameters increase. In this example the number of nodes in the group is held constant, and the percentage of messages delivered in the network is also held constant at 75%. This graph illustrates how modifying the  $B$  and  $k$  parameters causes linear increases in messages sent.

In conclusion the Random Ping, Gossip, and Aggressive protocols have  $O(n)$  behavior in a perfect network, and the Heartbeat and Ping protocols have  $O(n^2)$

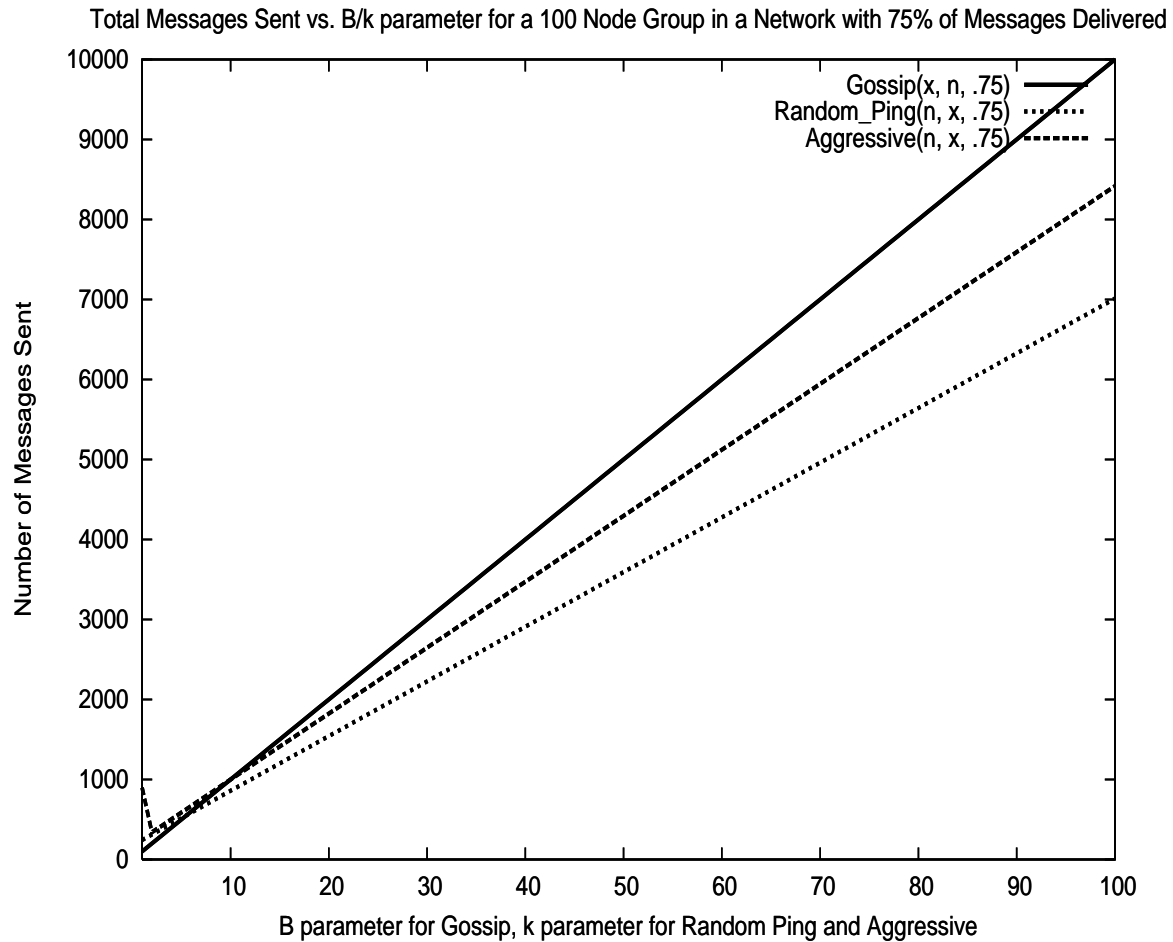


Figure 19: Number of Messages Sent vs.  $B, k$  Parameters

behavior in a perfect network. As the reliability of the network decreases, all of the protocols maintain their complexities except for the Aggressive protocol which becomes  $O(n^2)$ . Figure 20 summarizes these observations.

## 5.2 Accuracy versus Speed

There is a trade-off between the accuracy and speed properties of a failure detector. If an application demands faster detections of failures, then the obvious solution would be to increase the rate that messages are sent and decrease the timeout values. In a perfect network environment where packets are never dropped or delayed then this would be an ideal solution. Unfortunately this situation is a rarity. If packets are delayed or lost, a protocol may incorrectly detect a node as failed. So as detection speed increases, accuracy decreases.

| Number of Messages Sent |  |   |
|-------------------------|--|---|
| Protocol                | 100% of Messages Reach Their Destination | Unreliable Network Messages Sent Estimation |
| Heartbeat               | $n^2 - n$                                | $O(n^2)$                                    |
| Ping                    | $2(n^2 - n)$                             | $O(n^2)$                                    |
| Gossip                  | $Bn$                                     | $O(n)$                                      |
| Random Ping             | $2n$                                     | $O(n)$                                      |
| Aggressive              | $2n$                                     | $O(n^2)$                                    |

Figure 20: Protocol Message Complexities

The opposite holds as well. If an application requires the failure detector to be very accurate, then the protocol's timeout value should be increased, and the rate that messages are sent should be decreased. If the message delivery delay is high, then this should be an efficient solution. If packets are being dropped then an alternative solution is to increase the number of timeouts allowed before a node is added to the suspect list. If a node actually fails, then it will have to be pinged several times until it has breached the timeouts allowed barrier. As a result detection times will be higher.

Since each of the protocols can be configured to be fast or accurate it does not seem fair to claim that one protocol is 'the best'. From the previous discussion it would seem that 'the best' protocol is the one that can be configured to be fast or accurate with the least amount of negative side effects. Due to their inability to scale well it would appear that the Heartbeat and Ping protocols would have the largest number of side effects. Increasing the rate at which messages are sent is risky business when the protocol is sending  $O(n^2)$  messages per round. The Gossip protocol's ability to infect the group with information should allow it to detect failures quickly, with few detection errors. The Random Ping protocol should be accurate since it confirms a possible failure using the swarm procedure. Since it only pings one node per round,



it may not be able to detect failures as fast as the Gossip protocol. The Aggressive protocol's swarm procedure modifications should make it even more accurate than the Random Ping protocol. The burst procedure should make the protocol faster, but this property will deteriorate as the number of packets lost in the network increases. The tests presented in section 8 and discussed in section 9 will prove or disprove these hypotheses.

### 5.3 Complexity

A failure detection protocol should be designed so that it is lightweight and does not consume large amounts of resources or CPU time. All of the protocols in this thesis execute very simple algorithms. The CPU time needed to execute these procedures should be insignificant. The real burden that a failure detector places on the device using it comes from the time and resources needed to process messages both sent and received by the protocol. On a limited resource device such as a PDA, this can become a serious issue. Assuming algorithm computation is minimal then it makes sense to compare the protocols in terms of message complexity. This issue was investigated in section 5.1.

### 5.4 Summary

The focus of this thesis was to develop a new protocol based on Chandra, Goldszmidt, and Gupta's protocol that would be faster and more efficient. After examining each protocol in detail and comparing them, several hypotheses were formed. These hypotheses will be used when examining the test results presented in section 8.

1. Due to their  $O(n^2)$  message complexities, the Heartbeat and Ping protocols are inefficient compared to the Gossip, Random Ping, and Aggressive protocols.
2. In terms of speed, the Heartbeat, Ping, Gossip, and Aggressive protocols should be competitive, but the Random Ping protocol will be sub-par because of the random ping selection procedure.
3. The speed of the Aggressive protocol will deteriorate as the network becomes more unreliable.
4. The Random Ping and Aggressive protocols should maintain accuracy as network reliability decreases.
5. The Gossip protocol's only weakness appears to be large message sizes for large groups.
6. The Aggressive protocol should be fast and accurate, but these properties will deteriorate as network reliability decreases.

## 6 Protocol Implementations

The design for the protocol implementations is fairly simple. Common functionality required by all of the protocols is implemented in the `Device.java` abstract base class. The four protocols are implemented in subclasses derived from the `Device.java` base class. Figure 21 shows the design of the protocol classes. The `Device` abstract base class is at the top of the figure. The first set of methods in italics are declared `public abstract`, and each of the protocol subclass must provide an implementation of these methods. There is also the single `public` method, `getSuspects`, defined in the base class. The four protocol subclasses are shown below the `Device` base class.

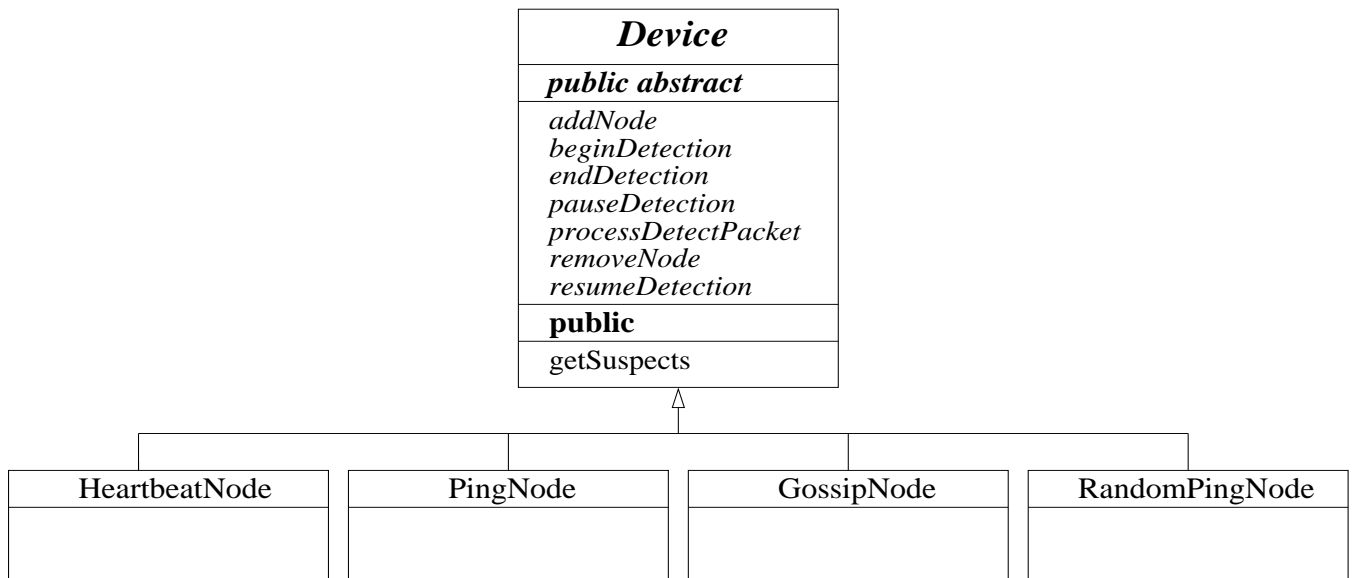


Figure 21: Protocol Class Structure

### 6.1 Abstract Device Base Class

The `Device.java` base class is the foundation building block of the failure detector components. It encapsulates the functionality that would be found in the upper layers of the application using the failure detector, as well as functionality for simulating network conditions.

#### 6.1.1 Communication

The main thread of the `Device.java` class continually receives and examines failure detector messages. Failure detector messages are encapsulated in UDP packets. If the incoming message is of a valid type, the thread calls a method to process the contents of the message. The method is defined as `abstract` in `Device.java`. Each of

the protocols will implement this method in their subclasses. A second method is defined for sending outgoing failure detector messages. The protocol subclasses can call this method and the message contents will be packed in UDP packets and sent to their destinations.

A second thread is created in the constructor of the `Device.java` class. This thread continually listens for TCP connections. TCP connections are used to communicate messages that must be received by every node. Example messages that require such reliable communication are requests from the testing component to add and remove nodes from the group, start the failure detector process, and requests for a node's suspect list. When events are generated at a node such as sleeping, joining, failing, or leaving the group, the node communicates with the testing component using TCP.

It is important to discuss the possibility of message fragmentation since UDP is used for transporting failure detector messages. If the size of the messages sent is large, message fragmentation may occur at the IP layer. If fragmentation occurs, there is the possibility that one of the fragments is not received on the other end and the entire message is lost. So for protocols that send large messages, especially the Gossip protocol, message fragmentation can have negative impacts. Since messages may be dropped due to lost fragments, less messages will reach their destinations, creating the possibility for incorrect detections. Message fragmentation was a variable that was not controlled in the test environment used in this thesis, and may have had an impact on the results of the tests conducted.

There is also the possibility that failure detector messages are larger than the maximum payload of a UDP packet which is 65,535 bytes. Several UDP packets may be needed to send a single failure detector message. Since UDP is an unreliable protocol, there is the possibility that a UDP packet may not reach its destination. So if the size of failure detector messages depends on the group size, performance may degrade as the group size increases. To avoid the negative aspects of message fragmentation and the need for multiple UDP packets per message, protocol adjustments may be needed. In the case of the Gossip protocol, each gossip message could be adjusted so that it only includes a portion of the node identifiers and counters. The rest of the list could be sent in later messages.

### **6.1.2 Event Generation**

Part of the functionality of the testing framework and failure detector components is the ability to specify the exact time when a node joins or leaves the group, sleeps for a given amount of time, or fails. These events are defined in the lifetime of a node in the input files used when a node is created. The input files will be discussed in a later section of this paper. Several internal classes are defined for each of the possible

events. A timer is used to schedule execution of the events at the time specified in the input file.

### **6.1.3 Suspect List**

A HashMap is used in the Device.java class to keep track of the nodes that the failure detector suspects as failed. The HashMap takes integer node identifiers and maps them to detection times. The testing component can make a request for a node's suspect list. It can also include an identifier argument. If the identifier argument is included the node periodically checks the HashMap for the identifier. If it is in the map, then the suspect list is sent to the testing component. If the identifier is not in the map, then the node waits and continually checks the map until the node has detected the node. Once it appears in the suspect list, the node sends the list to the testing component.

### **6.1.4 Simulation Functionality**

A section of the input files defines the characteristics of the host device that the failure detector is running on. Some of the characteristics include the probability that a packet is correctly processed by the device, or whether or not a packet is sent correctly from the device. Methods are defined in the Device.java class to simulate these characteristics using the parameters defined in the input files. The percentage of packets dropped in the network is also an input file parameter. A method is defined in Device.java to simulate whether or not a packet is dropped. This method is called prior to sending a failure detection packet. The return value of the method determines whether or not the packet is sent.

## **6.2 Design of Protocol Subclasses**

Each of the protocols except for the Aggressive protocol were implemented in subclasses of the Device base class. The Aggressive protocol features are built into the RandomPingNode class. They are turned on with a constructor parameter. Each subclass provides implementations of the methods used to start, stop, and pause the failure detection procedures. They also provide implementations of the method called to process incoming failure detection packets.

Each of the protocols perform some kind of task over and over again. For the Heartbeat protocol the task is to send heartbeat messages to all group members. For the Gossip protocol the task is to send the list of identifiers and counters to a few other randomly selected members. Each protocol class uses timers to execute their unique tasks. The tasks are executed at periodic intervals defined by an input file parameter. Since each protocol class is an instance of a Device, the main thread of each node is the main thread of the Device.java class that processes incoming UDP failure

detection packets. So separate threads are used to execute the tasks required by each protocol. Since the threads would be accessing common data structures, synchronization mechanisms were used where concurrent access was a possibility.

To add a new protocol to the framework several things need to be done. One task is to implement the protocol in a subclass of `Device.java`. The next task is to adjust the `Defs.java` file so that it includes the new protocol as well as the parameter names that would be used in input files. The `Defs.java` class is discussed in the input file section. The last step is to add a method in the `Setup` class for creating instances of the new protocol class.

## 7 Test Framework

The testing framework consists of four main components. The components are input files, `Parser.java`, `Tester.java`, and `Setup.java`. The flexibility of the testing framework comes from the input files used.

### 7.1 Input Files

Consider the situation where one has implemented all of the protocols that were discussed in previous sections of this paper. The next step is to test the protocols and see how they perform in different environment conditions. It would be useful to have the ability to vary many conditions including the loss rate of the network, the protocol used, the protocol parameters, the conditions of the host running the application, the number of nodes in the group, and the events that take place for each node. This flexibility was the driving force behind the input files used in this thesis.

#### 7.1.1 Example Input File

Suppose one wants to test the Ping protocol in a very simple environment with only two nodes. The network should be fairly reliable. One node will run on a host that is more stable than the other. The nodes will join the group at the same time, but they will have sleep events at different times. Only one of the nodes will fail. This situation could be easily organized into an input file. An example of such an input file is as follows.

Each input file should always have a version. The version is useful for keeping track of tests with their corresponding input files.

```
VERSION = Test3a_Wed_Sep_24_13:14:34_EDT_2003
```

The next task is to specify the protocol being tested, as well as the parameters for

the protocol.

```
PROTOCOL_TO_TEST = Ping
PING_ACK_TIMEOUT = 200
PING_SEND_DELAY = 400
```

Another important parameter to include is the reliability of the network. This is done by specifying the probability of a packet being dropped.

```
PACKET_DROPPED_PROBABILITY = 0.05
```

Now that the protocol and network have been specified, the next step is to define the host devices of the two nodes in the test. The first definition is for DeviceX.

```
TYPE = DeviceX
PACKET_PROCESSED_PROBABILITY = 0.93
PROCESSING_DELAY_TIME_MIN = 10
PROCESSING_DELAY_TIME_MAX = 500
BUSY_SLEEP_PROBABILITY = 0.15
SLEEP_TIME_MIN = 50
SLEEP_TIME_MAX = 1000
PACKET_SENT_PROBABILITY = 0.94
END_TYPE
```

The second device definition should have most of the same values as the first device definition. This is done by extending the DeviceX definition using the 'EXTENDS' keyword. The device parameters that need to be different are changed by overriding the values in the base definition with new values. In this case DeviceY is more reliable than DeviceX because of its higher processing and sending probabilities. DeviceY is also less likely to be in a sleep or busy state.

```
TYPE = DeviceY EXTENDS DeviceX
PACKET_PROCESSED_PROBABILITY = 0.98
BUSY_SLEEP_PROBABILITY = 0.07
PACKET_SENT_PROBABILITY = 0.99
END_TYPE
```

The final task is to define the node lifetimes for each of the nodes in the group. The first definition will be for the node with identifier equal to 1, which is running on device type DeviceX. The time values on the left correspond to the number of milliseconds that have passed since the node was created. In this case node 1 joins the group one second after it is created, sleeps for two seconds once five seconds have

passed since it was created, and fails fifteen seconds after it was created.

```
LIFETIME : TYPE = DeviceX : ID = 1
1000 : JOIN
5000 : SLEEP : 2000
15000 : FAIL
END_LIFETIME
```

The second node's identifier is equal to 2 and it is running on device type DeviceY. Node 2's definition can automatically have the same events as node 1 by extending node 1's definition. The second node adds a second sleep event with a duration of one second. For this test, only the first node is supposed to fail. So the fail event is deleted from the base node definition by specifying the time of the event and using the 'DELETE' keyword.

```
LIFETIME : TYPE = DeviceY : ID = 2 : EXTENDS 1
3000 : SLEEP : 1000
15000 : DELETE
END_LIFETIME
```

All that is left to do is execute the test program using the input file definition as a command line argument. The testing component is discussed in a later section of this paper.

### 7.1.2 Grammar

The example input file presented in the previous section highlights many of the capabilities of the input file language, but more features are available. A formal grammar definition of the input file language is as follows.

```
<input file> → <version>
               <definition>+
```

```
<version> → 'VERSION' '=' <string>
```

```
<definition> → <import> | <debug level> | <protocol name> | <parameter>
               | <device definition> | <node lifetime>
```

```
<import> → 'IMPORT' <string>
```

```
<debug level> → 'DEBUG_LEVEL' '=' <integer>
```

*<protocol name>* → 'PROTOCOL\_TO\_TEST' '=' *<protocol type>*  
*<protocol type>* → 'Heartbeat' | 'Ping' | 'Gossip' | 'Randomized' | 'Aggressive'

*<parameter>* → *<protocol parameter>* | *<network parameter>*

*<protocol parameter>* → *<heartbeat parameter>* | *<ping parameter>*  
 | *<gossip parameter>* | *<random parameter>*  
 | *<aggressive parameter>*

*<heartbeat parameter>* → *<heartbeat parameter type>* '=' *<integer>*  
*<heartbeat parameter type>* → 'HEARTBEAT\_TIMEOUT' | 'HB\_CHECK\_DELAY'  
 | 'HB\_SEND\_DELAY'

*<ping parameter>* → *<ping parameter type>* '=' *<integer>*  
*<ping parameter type>* → 'PING\_ACK\_TIMEOUT' | 'PING\_SEND\_DELAY'

*<gossip parameter>* → *<gossip parameter type>* '=' *<integer>*  
*<gossip parameter type>* → 'GOSSIP\_TIMEOUT' | 'B' | 'GOSSIP\_CHECK\_DELAY'  
 | 'GOSSIP\_SEND\_DELAY'

*<random parameter>* → *<random parameter type>* '=' *<integer>*  
*<random parameter type>* → 'TIMEOUT1' | 'TIMEOUT2' | 'K' | 'RANDOM\_PING\_SEND\_DELAY'

*<aggressive parameter>* → *<aggressive parameter type>* '=' *<integer>*  
*<aggressive parameter type>* → 'TIMEOUT1' | 'TIMEOUT2' | 'K' | 'PING\_SEND\_DELAY'  
 | 'TRACK\_BEST' | 'TRACK\_WORST'  
 | 'WORST\_NODES\_SEND\_DELAY' | 'GOSSIP\_COUNT'

*<network parameter>* → *<drop probability>*  
*<drop probability>* → 'PACKET\_DROPPED\_PROBABILITY' '=' *<double>*

*<device definition>* → 'TYPE' '=' *<string>* [ 'EXTENDS' *<string>* ]  
*<device parameter>*\*  
 'END\_TYPE'

*<device parameter>* → *<double device parameter>* | *<integer device parameter>*  
*<double device parameter>* → *<double device parameter type>* '=' *<double>*  
*<double device parameter type>* → 'PACKET\_PROCESSED\_PROBABILITY'  
 | 'BUSY\_SLEEP\_PROBABILITY'  
 | 'PACKET\_SENT\_PROBABILITY'

*<integer device parameter>* → *<integer device parameter type>* '=' *<integer>*  
*<integer device parameter type>* → 'PROCESSING\_DELAY\_TIME\_MIN'



```

| 'PROCESSING_DELAY_TIME_MAX'
| 'BUSY_SLEEP_TIME_MIN' | 'BUSY_SLEEP_TIME_MAX'

<node lifetime> → 'LIFETIME' ':' 'TYPE' '=' <string> ':' 'ID' '=' <integer>
                [ ':' 'EXTENDS' <integer> ]
                <event>+
                'END_LIFETIME'

<event> → <integer> ':' <event type>
<event type> → 'JOIN' | 'LEAVE' | 'FAIL' | 'DELETE' | <sleep event>
<sleep event> → 'SLEEP' ':' <integer>

<double> → <digit>+ "." <digit>+ | <integer>
<integer> → <digit>+
<digit> → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<string> → Any Legal Java String

```

### 7.1.3 Import Statement

The 'IMPORT' keyword is used in the input files similar to the way '#include' and 'import' statements are used in C/C++ and Java. When the Parser encounters an import statement it opens the specified file if it exists and processes it like any other input file, except that it adds the specified information to the existing data structures instead of new ones. Once the imported file has been processed, the Parser continues where it left off in the original input file. The Parser class is covered in more detail in a later section of this paper.

Frequently used input file sections can be stored in separate files and then imported as necessary. Using imported files is also useful when certain conditions need to be held constant, but others need to vary. The conditions that are being tested are easier to find in the main input file, and less space is taken up by conditions that are being held constant.

### 7.1.4 Input File Semantics

An input file may be syntactically correct, but at the same time be invalid. There are several conditions that must be checked to make sure an input file is valid.

1. When a device type is declared, the name specified cannot already be used in another device type definition.
2. When using 'EXTENDS' in a device type definition, the base device type name given must already be defined. In the example input file presented earlier DeviceY extends DeviceX.

```
TYPE = DeviceY EXTENDS DeviceX
```

DeviceX must be defined in an earlier section of the input file, or the input file is invalid.

3. In a node lifetime declaration, the device type specified must be defined somewhere in the input file. This condition is checked for every node lifetime declaration after the Parser has parsed the input file. In the example input file presented earlier node 1 is of type DeviceX.

```
LIFETIME : TYPE = DeviceX : ID = 1
```

If DeviceX is not defined at some point in the input file, the input file is invalid.

4. If the identifier specified in a node lifetime definition has already been used, a warning is issued. The assumption is that the user may want to override a previously defined node, whether it be in the current file or an imported file.
5. When using 'EXTENDS' in a node lifetime definition, the base node identifier must already be defined. In the example input file presented earlier node 2 extends node 1.

```
LIFETIME : TYPE = DeviceY : ID = 2 : EXTENDS 1
```

In this case the identifier 1 must be the identifier of a previously defined node lifetime, or the input file is invalid.

6. A warning is issued if no event exists for the time value specified when using 'DELETE'. In the example presented earlier the failure event at time 15000 is deleted in node 2's lifetime declaration.

```
15000 : DELETE
```

If no event had been present with starting time 15000 in node 1's definition, then a warning would have been issued.

7. A warning is issued if the time of a declared event is already the starting time for a previously declared event, whether in the current definition or a base node lifetime definition.

### 7.1.5 Keyword and Symbol Adjustments

The keywords and symbols used in the input language are defined in the source file Defs.java. This file is also where the protocol names and protocol parameters are

defined. To change any of the keywords or symbols in the language, all that needs to be done is alter the definitions in Defs.java. To change the actual syntax of the language the Parser class would have to be altered. In summary the Defs.java file provides flexibility in terms of the keywords and symbols used in an input file, but to change the language in terms of syntax and semantics, the Parser file would have to be adjusted.

## 7.2 Parser

The Parser class is used to read and process input files. It is responsible for checking syntax, as well as validate node lifetime and device type definitions. If the Parser encounters an error it will print a message describing the problem encountered and the line number in the input file where the error was encountered. If the input file is valid, the device type definitions, node definitions, protocol parameters, and network parameters are stored in data structures that can be accessed using several public methods. These methods are:

- `getDefs()`  
This method returns a HashMap object containing information such as the protocol being tested and the debug level.
- `getNodes()`  
This method returns a HashMap object that maps node identifiers to NodeDef objects. NodeDef is a class used to store all the information needed to create one of the failure detector protocol classes. The NodeDef object contains methods to access the device parameters for the node and the events in the node's lifetime.
- `summary()`  
This public method prints details of all the parameters, devices, and nodes defined in the input file. The summary information is printed to standard output and is useful for debugging an input file.

The Tester and Setup classes use `getDefs()` and `getNodes()` to access the input file definitions and create the failure detector node objects. Figure 22 shows how the Tester and Setup classes use a Parser object to process input files. It also shows how a Parser object creates NodeDef objects that contain a collection of events in the node's lifetime. The Tester and Setup classes will be discussed in the next section.

## 7.3 Tester and Setup Classes

The Tester class is responsible for creating nodes and collecting test data. The Setup class is similar to the Tester class, except that it does not have the data collection functionality. The Tester class extends the Setup base class, adding functionality for group organization and data collection.

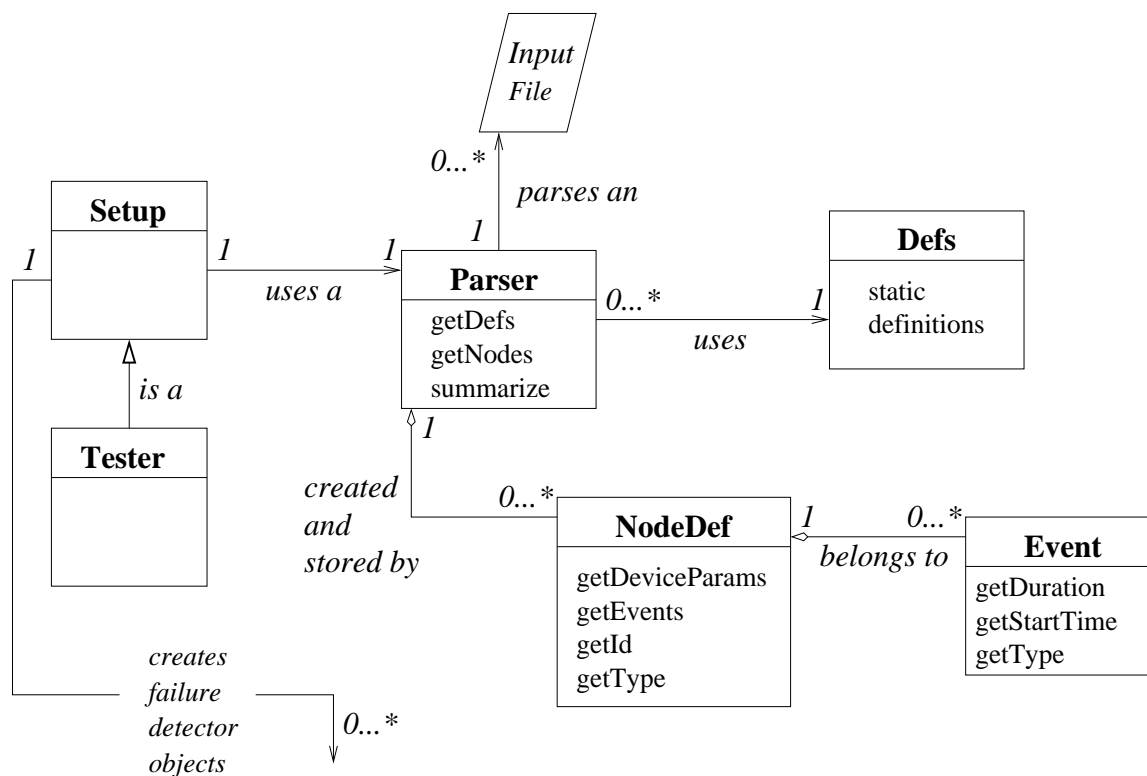


Figure 22: Test Framework Class Interaction

### 7.3.1 Setup

The Setup class is used to read and parse an input file, and create the nodes defined in that file. The Setup class takes two arguments. The first argument is the name of the input file to read. The second argument is the host name of the computer that the Tester class is running on. The host name argument is needed so that nodes know where to send messages relating to events such as joining and leaving the group, failing, and suspect list information. Once the nodes have been created, the Setup class has no other responsibilities. A consequence of the Setup class' design is that it does not terminate on its own. To overcome this obstacle the scripts used in the Tests section executed a separate script that killed any Java processes belonging to the user.

### 7.3.2 Tester

The Tester class extends the Setup class and has all the functionality of the Setup class as well as other methods needed for group organization and data collection. The Tester class takes the total number of nodes that will be in the group as an additional parameter. This argument is needed since only a few nodes may be created by the Tester process. The rest would be created on other host computers in the network.

Nodes contact the Tester component to notify it when they join or leave the group, and also when they fail. It is then the Tester's responsibility to act on these notifications. When a node joins or leaves the group, the Tester records the host name and port that the node will be receiving failure detection packets at. It also needs to notify all the other nodes that are already in the group about the new node. It sends the host and port information of the new node to all of the nodes already in the group, and also sends the host and port information of the nodes already in the group to the new node. If a node leaves the group, the Tester needs to notify all the nodes about the leaving member. When a node fails, the Tester instance has more complex responsibilities.

If a node notifies the Tester that it is about to fail, the Tester executes several tasks. The first is to remove it from its data structure storing information about the current nodes in the group. The second task is to record the time that the node failed. The last task is to set a timer to print summary data. Before the summary can be printed, the Tester sends messages to all the nodes asking them to send their suspect lists. It also includes the identifier of the failed node. Once all of the suspect lists have been received it prints a summary report of the detection times. An example of a minimal summary report is as follows.

```
# min, max, avg, detectns, nodes, replies, msgs, bytes, avg bytes/msg, errors
# RANDOM_PING_SEND_DELAY, K, TIMEOUT1, TIMEOUT2
229 40190 8195.0 100 100 100 142168 990251 6.965 0 75 3 30 30
```

The first two lines identify the columns of the third line. The columns in this report are the minimum, maximum, and average detection times, the number of nodes that correctly detected the node failure, the number of nodes in the group, the number of replies to the suspect list request, the total number of failure detection messages sent by the nodes, the total number of bytes sent, the average bytes per message sent, the number of errors (non-failed nodes reported as failed), the random ping send delay, the  $k$  value used for the swarm procedure, and finally the timeout values used in the Random Ping protocol. A more detailed report can be obtained depending on the debug level used. The detailed report includes full listings of every failed node with the node identifiers and the time they detected the failure. The summary also includes any nodes that were incorrectly reported as failed along with the identifiers of the nodes that reported them.

Since many tests were run in a script file, the Tester class also has the functionality to exit after printing a summary report. The scripts also needed to redirect the output of the Tester class since it prints to the standard output.

Figure 23 shows the interaction of the Tester and Setup classes with failure detector nodes. In the example a Setup object creates nodes 1 and 2. A Tester object on another host creates a node with identifier 3. The next event is node 1 notifying the Tester that it is joining the group. Node 2 joins next, and the Tester object notifies node 1 to add node 2. The Tester also notifies node 2 to add node 1. Node 3 joins the group next. The Tester notifies all the nodes about the other nodes in the group. The next event is node 2 notifying the Tester that it is about to fail. A short time later the Tester object sends messages to nodes 1 and 3 requesting their suspect lists. The two nodes respond with their suspect lists. The Tester object prints a report of the detection times and then exits. Since node 3 was running in the same process as the Tester object it exits as well. The process running node 1 and the Setup object on the other host were killed.

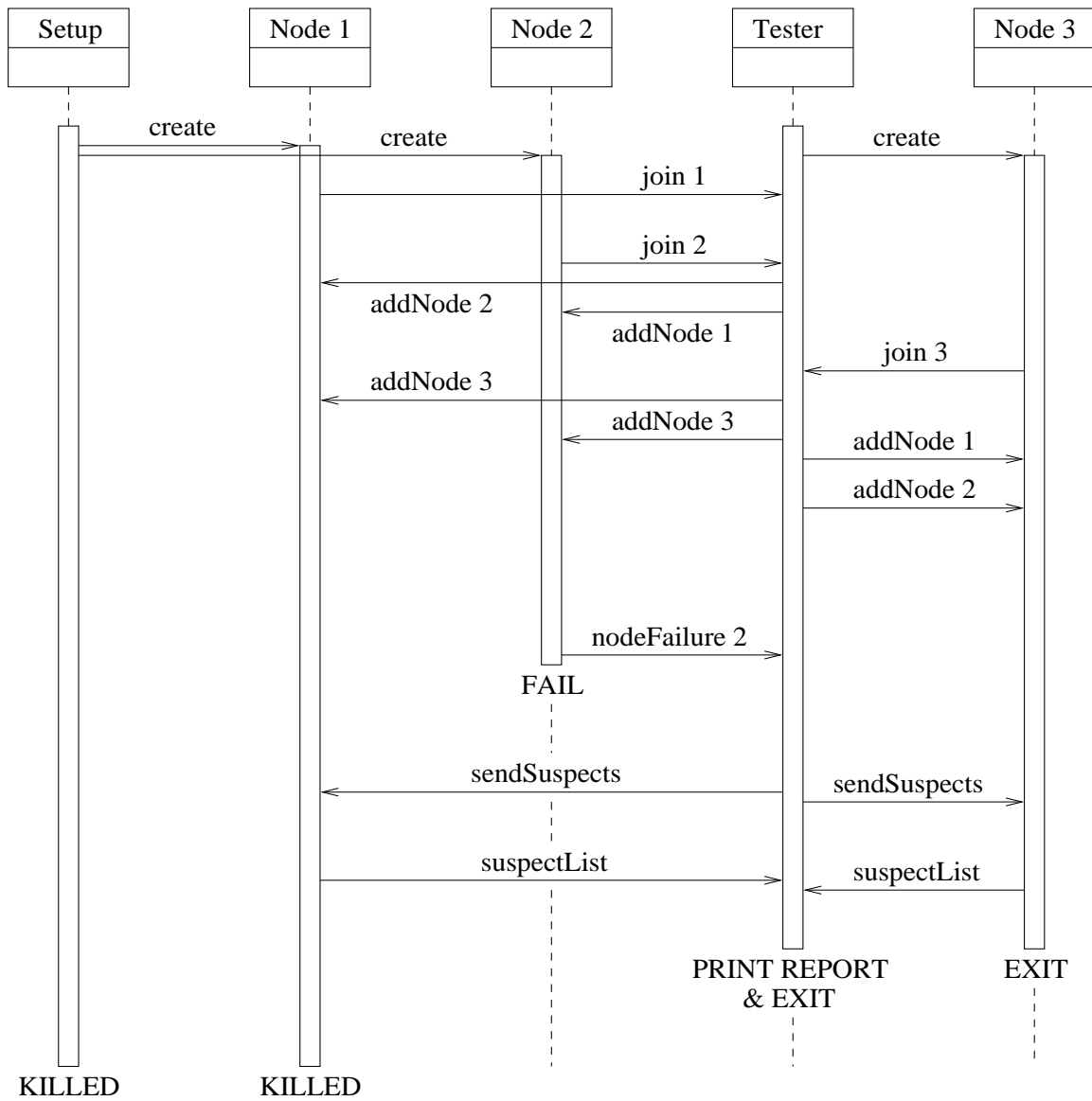


Figure 23: Tester and Setup Interaction with Failure Detector Objects

## 8 Tests and Results

The test environment provides four main variables for testing. These variables are the protocol parameters, the number of nodes in the group, the reliability of the network, and the reliability of the device each node runs on. The testing strategy was to vary each of these variables individually while holding the others constant. This strategy results in a set of protocol parameters for a given environment, each protocol's performance as group size increases for a configured set of protocol parameters,

each protocol's reaction to decreased network reliability, and finally each protocol's performance in an environment where node performance capabilities are unequal.

All of the tests in this section were conducted on Sun Workstations in Rochester Institute of Technology's computer science department. For consistency the same set of machines were used whenever possible based on availability and consideration for colleagues.

## 8.1 Protocol Parameters

The goal of the first test was to find a set of parameters for each protocol that could be used in future experiments. This was done by holding the number of nodes in the group constant at 100, the network delivery percentage at 100%, and each device with 100% message processing and sending capabilities. Scripts were written that looped through a series of parameters for each protocol. The data was then sorted by the number of errors and average detection times. A set of parameters was then chosen based on the sorted data. This set of parameters was used in future experiments where the protocol parameters needed to be held constant. The data for each of the protocols is presented in appendix A. The resulting set of parameters is as follows.

- Heartbeat  
HB\_SEND\_DELAY = 650  
HEARTBEAT\_TIMEOUT = 700  
HB\_CHECK\_DELAY = 700
- Ping  
PING\_SEND\_DELAY = 1000  
PING\_ACK\_TIMEOUT = 400
- Gossip  
GOSSIP\_SEND\_DELAY = 100  
GOSSIP\_TIMEOUT = 450  
B = 4  
GOSSIP\_CHECK\_DELAY = 450
- Random Ping  
RANDOM\_PING\_SEND\_DELAY = 70  
K = 4  
TIMEOUT1 = 20  
TIMEOUT2 = 30
- Aggressive  
RANDOM\_PING\_SEND\_DELAY = 100  
K = 4



```
TIMEOUT1 = 20
TIMEOUT2 = 60
```

It is not possible to conclude at this time whether or not these parameters are the optimal set for each protocol. Finding the optimal set of parameters would require running each protocol with every possible combination of parameters. In a simulated environment each test could be carried out quickly without having to coordinate processes on different workstations. Since running a test for every combination of parameters was not possible with the test framework used, the ones chosen here can only be considered an approximation of the optimal set of parameters.

## 8.2 Group Sizes

The second test's purpose was to see how each protocol performed over a range of group sizes. The protocol parameters found in the previous test were held constant over group sizes ranging from 10 nodes up to 200 nodes. The network reliability was held constant at 100% message delivery and each node device was held constant at 100% message processing and sending capabilities. This test would determine how important it is for the protocols to be dynamic. Dynamic in this context means the protocols will need to adjust their parameters on the fly as the number of nodes in the group increases or decreases. If implementing the protocols to be dynamic is not an option, then it may be the case that the upper layer of the application will need to monitor the failure detector and make protocol parameter adjustments in response to group size changes.

The data used to generate the following graphs is presented in appendix B.

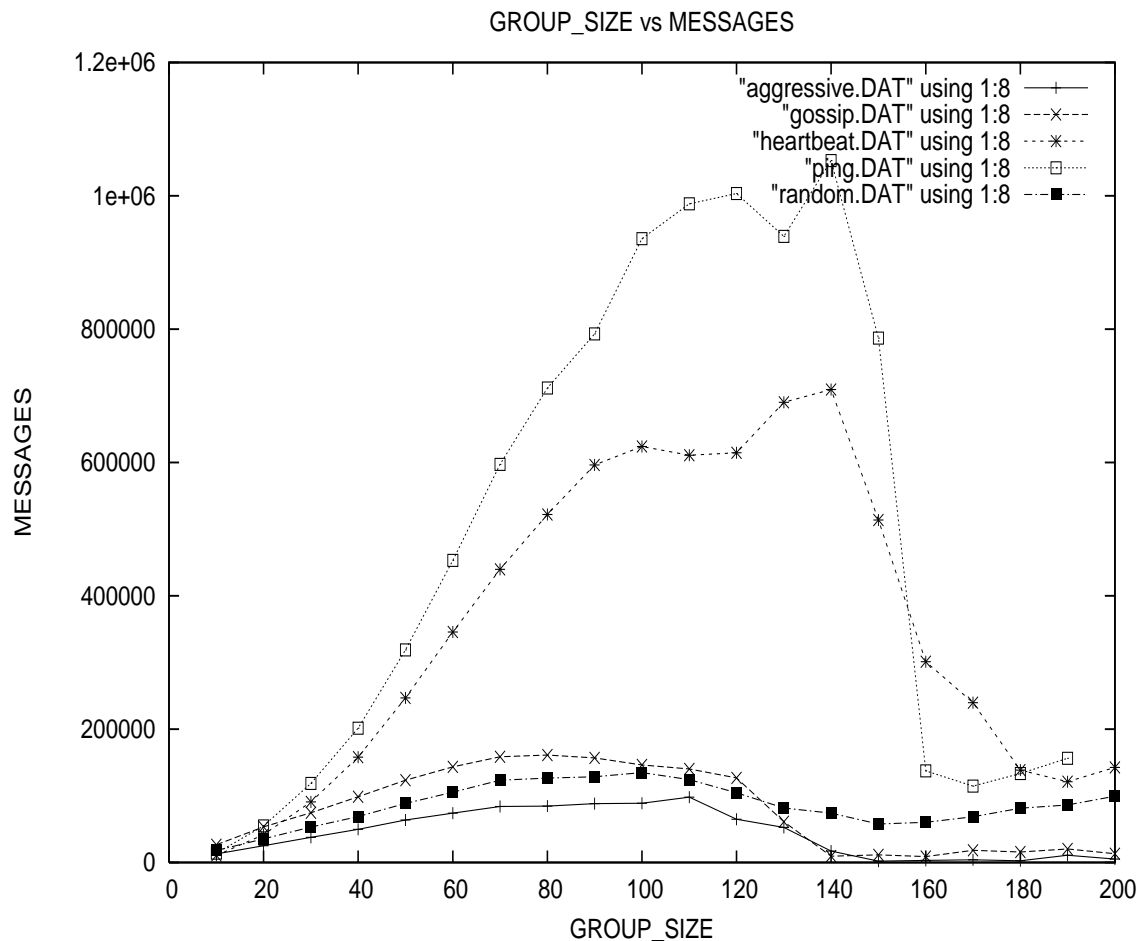


Figure 24: Group Size vs. Messages Sent

Figure 24 shows the number of messages sent by each protocol as the number of nodes in the group increases. The graph shows the  $O(n^2)$  message complexity of the Heartbeat and Ping protocols. This graph supports Hypothesis 1, since the other protocols, especially the gossip protocol, have as good or better detection times than the Heartbeat and Ping protocols using less messages. The exception for this is the Random Ping protocol, which has high average and maximum detection times. Graphs for average and maximum detection times are shown in figures 26 and 27. The graph also shows that as the group size grows beyond 100 to 120 nodes the protocols actually send less messages. The decrease in messages is caused by the protocols' inability to handle the increase in nodes. In summary the graph shows that unless the protocol is implemented to be dynamic, the nodes would have to be reconfigured as the group size changes.

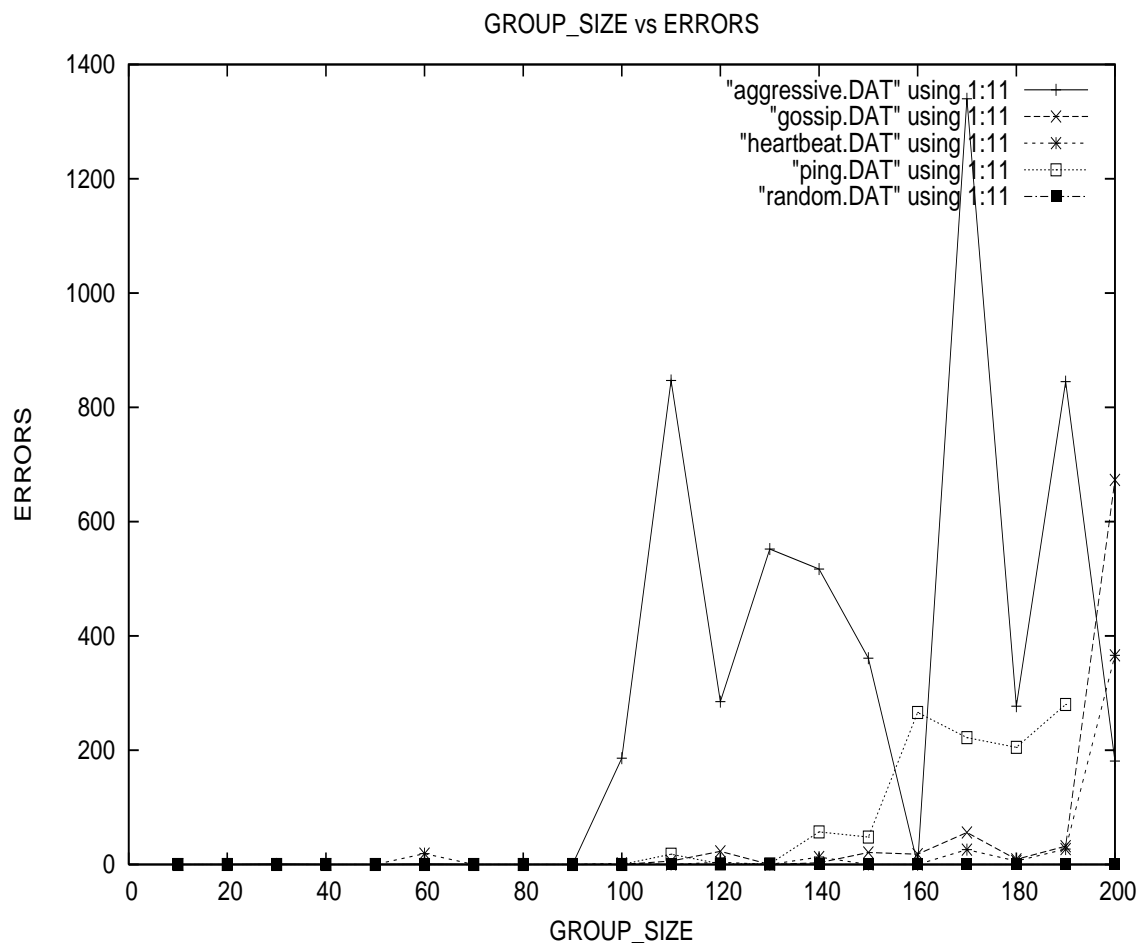


Figure 25: Group Size vs. Detection Errors

Figure 25 shows the total number of false detections made by the nodes as the number of nodes in the group increases. This graph shows that the Aggressive protocol is not accurate for group sizes larger than it is configured for. It also shows that the protocol will either need to be dynamic or be reconfigured as the group size changes, otherwise the modifications will not improve the accuracy of the Random Ping protocol, but in fact make it less accurate.

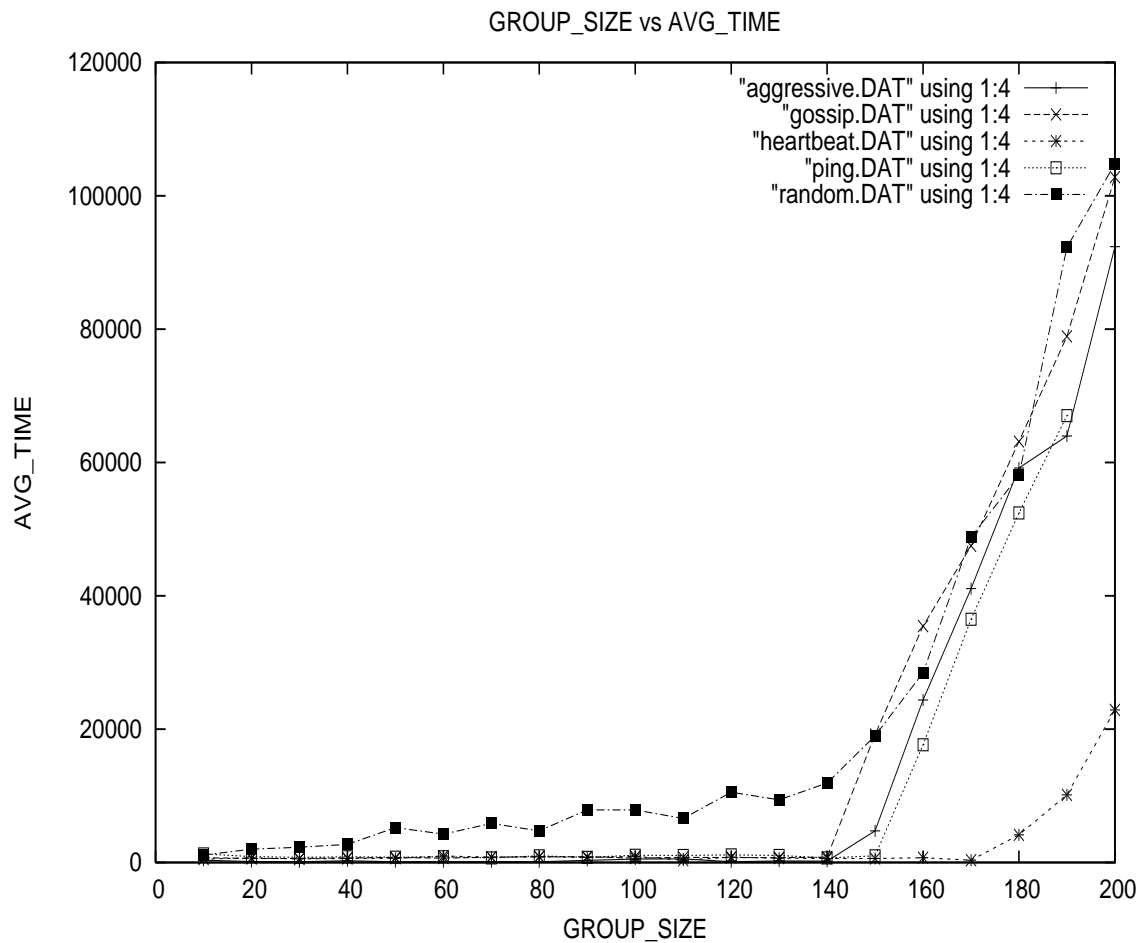


Figure 26: Group Size vs. Average Detection Time

Figure 26 shows the average time needed by a node in the group to detect a node failure as the number of nodes in the group grows. This graph supports Hypothesis 2 and shows how the Random Ping protocol has a higher average detection time compared to the other four protocols. The graph also shows that the average detection times increase for all of the protocols past a group size of 140 nodes. This means that the protocol parameters hold up for larger groups but only up to a certain point before they will need to be reconfigured.

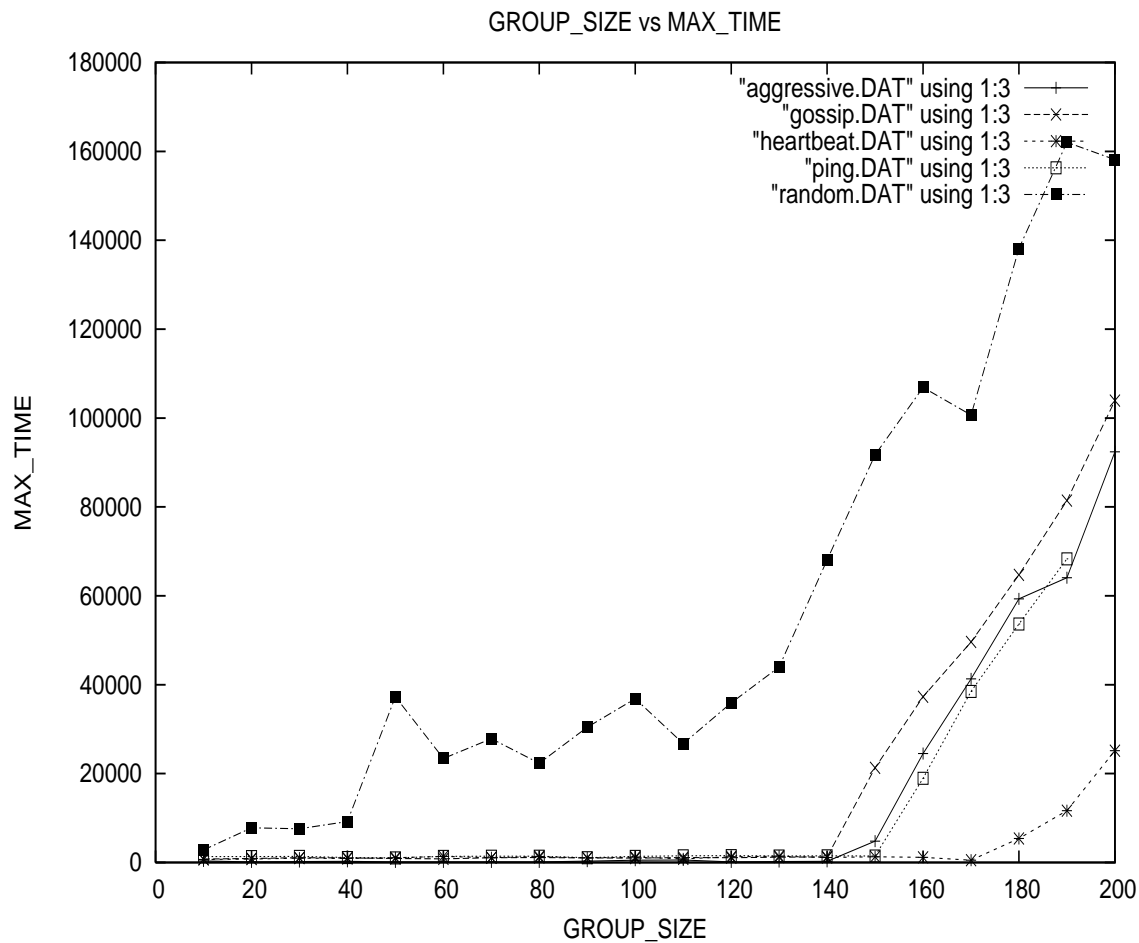


Figure 27: Group Size vs. Maximum Detection Time

Figure 27 is similar to figure 26 and also supports Hypothesis 2. In this graph the maximum detection time is plotted instead of the average detection time. This graph shows how the Random Ping protocol has a high maximum detection time due to the randomness of its ping procedure. The rest of the protocols have similar maximum detection times up to 140 nodes where they start to increase and need to be reconfigured.

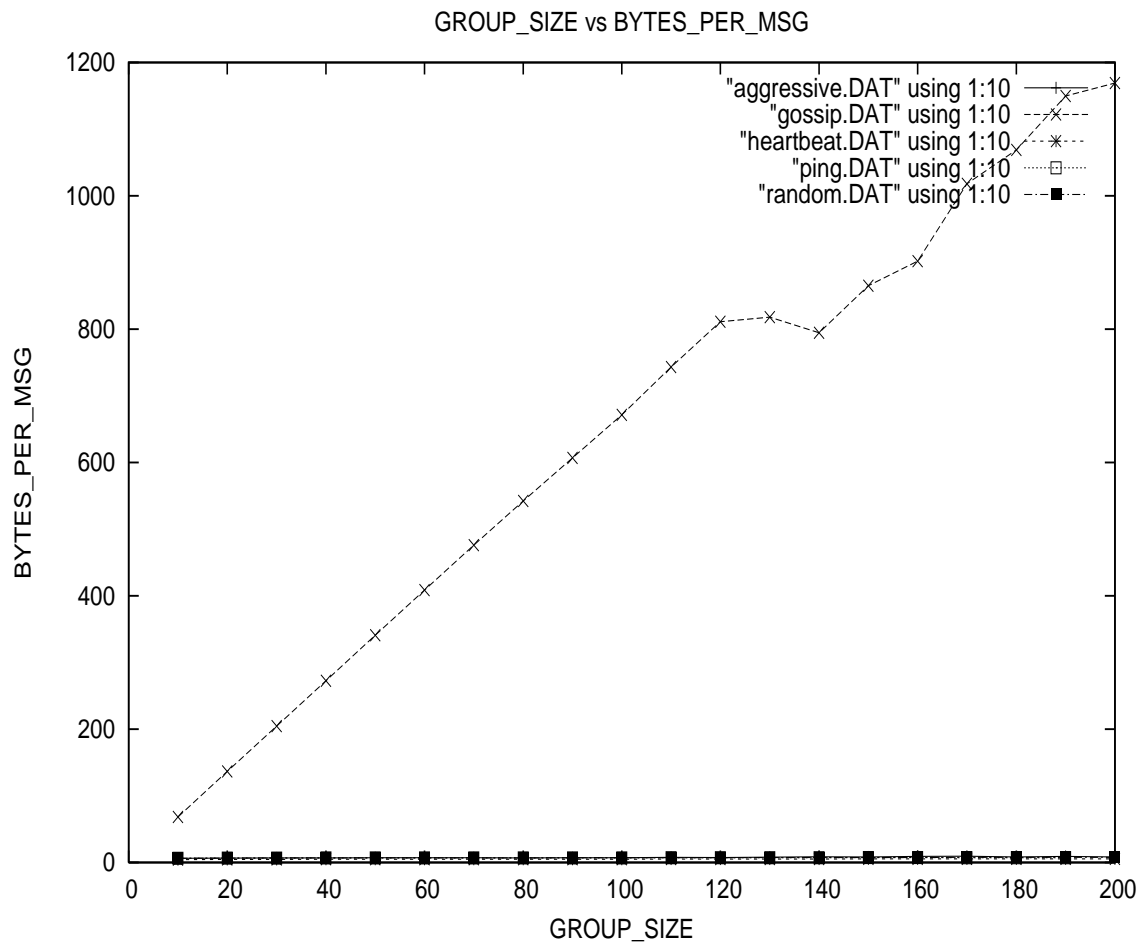


Figure 28: Group Size vs. Bytes Sent Per Message

Figure 28 plots the number of bytes per message sent for each of the protocols as the number of nodes in the group increases. The graph shows how the number of bytes per message sent for the Gossip protocol increases linearly with the number of nodes in the group, while remaining almost constant for the other four protocols. The fluctuations for the Gossip plot may have been caused by fragmentation. The graph supports Hypothesis 5 and shows how message size may be a weakness for the Gossip protocol and limit its ability to scale to large group sizes.

### 8.3 Network Reliability

The goal of the third test was to see how a protocol's performance changes as the reliability of the network decreases. For this test the protocol parameters from the first test were used and held constant. The number of nodes in the group was held constant at 100. Each node's device was configured for 100% message processing and sending capabilities. The percentage of packets delivered by the network was varied from 100% to 0%. A network component such as a router or switch may become overloaded or go down entirely causing a major change in network conditions. This test would show how each protocol would perform in the face of such a network environment change.

The data used to generate the following graphs is presented in appendix C.

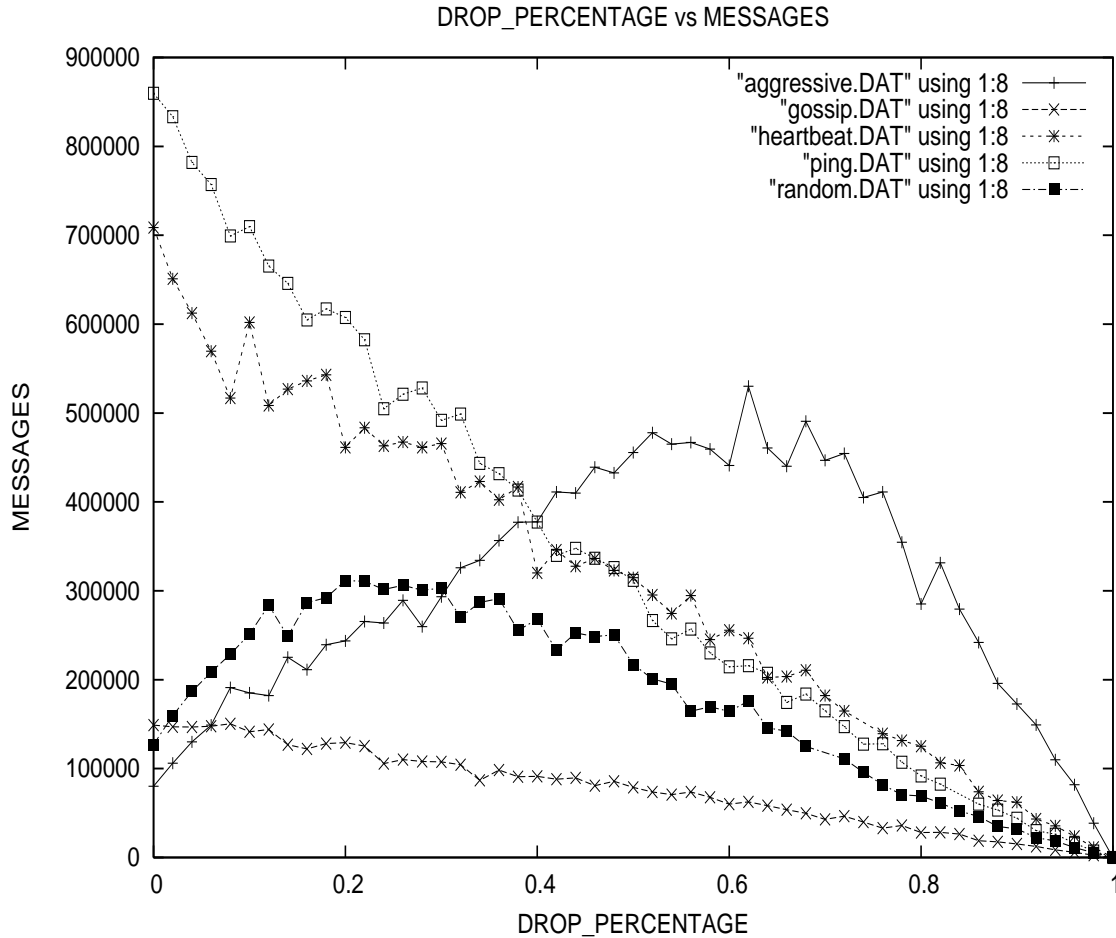


Figure 29: Network Delivery Percentage vs. Messages Sent

Figure 29 shows the total number of messages sent by the protocols as the number of messages dropped in the network increases. The graph shows how the Random Ping and Aggressive protocols send more messages as the network drop percentage increases. This is due to the swarm procedure. The Aggressive protocol sends more messages than the Random Ping protocol due to the burst procedure. The modifications implemented in the Aggressive protocol actually cause the protocol to send more messages than the other four protocols. With  $k$  equal to 4, the Aggressive protocol sends more messages than the other four protocols after a network drop rate of 40%. For network drop percentages less than 30%, the Aggressive protocol is competitive with the Gossip and Random Ping protocols in terms of messages sent. For drop percentages less than 40% the graph supports Hypothesis 1, since the other protocols, excluding the Random Ping protocol, obtain competitive detection times using less messages. See figures 31 and 32 for detection time comparisons of the protocols.



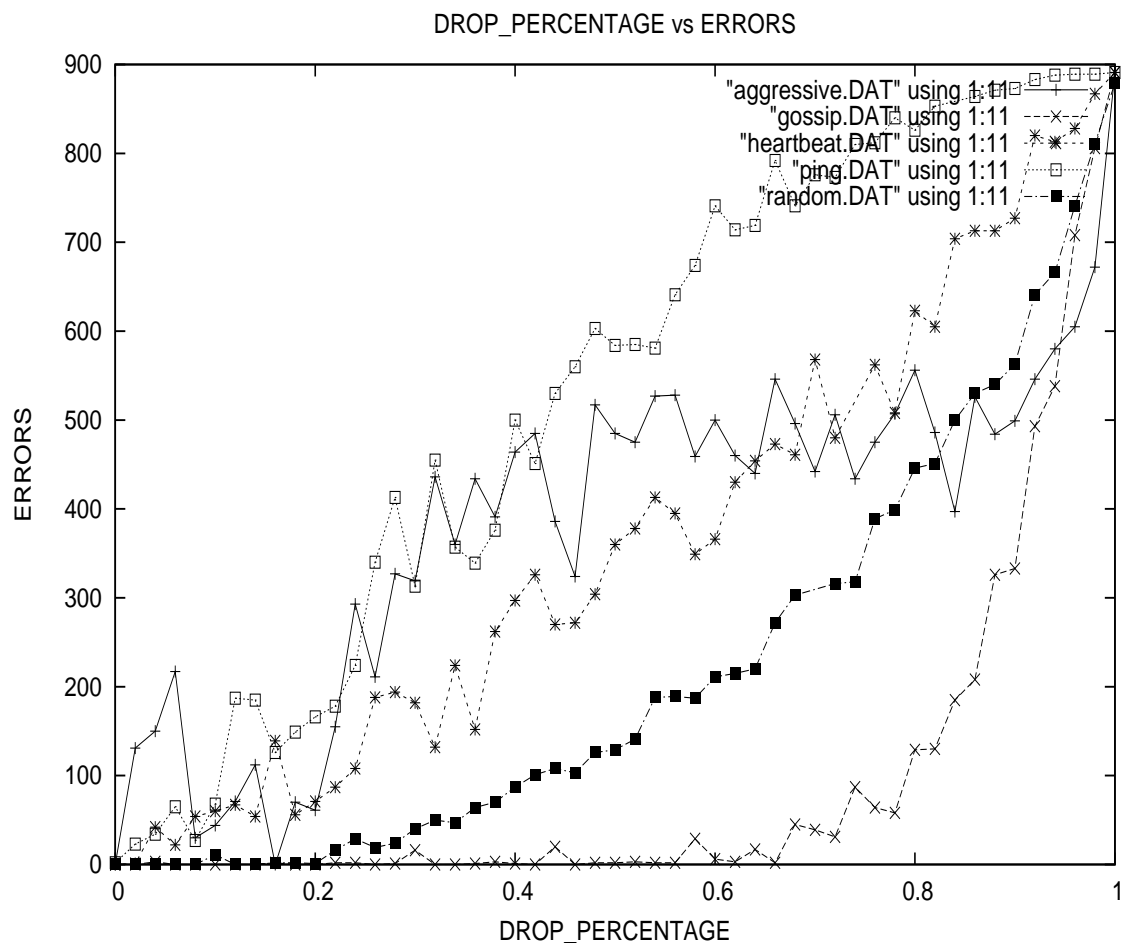


Figure 30: Network Delivery Percentage vs. Detection Errors

Figure 30 shows the total false number of detections made by the nodes in the group as the number of nodes in the group increases. The accuracy of each of the protocols deteriorates with the network packet drop percentage. The graph shows that the Gossip protocol is the most accurate, followed by the Random Ping protocol. The graph also shows that the modifications implemented in the Aggressive protocol do not improve the accuracy of the Random Ping protocol. Figure 30 disproves Hypothesis 4 since both the Aggressive and Random Ping protocols lose accuracy as the network reliability decreases. The graph supports Hypothesis 6 in that the accuracy of the Aggressive protocol deteriorates with the network reliability although it was not predicted that the accuracy of the Aggressive protocol would deteriorate so quickly.

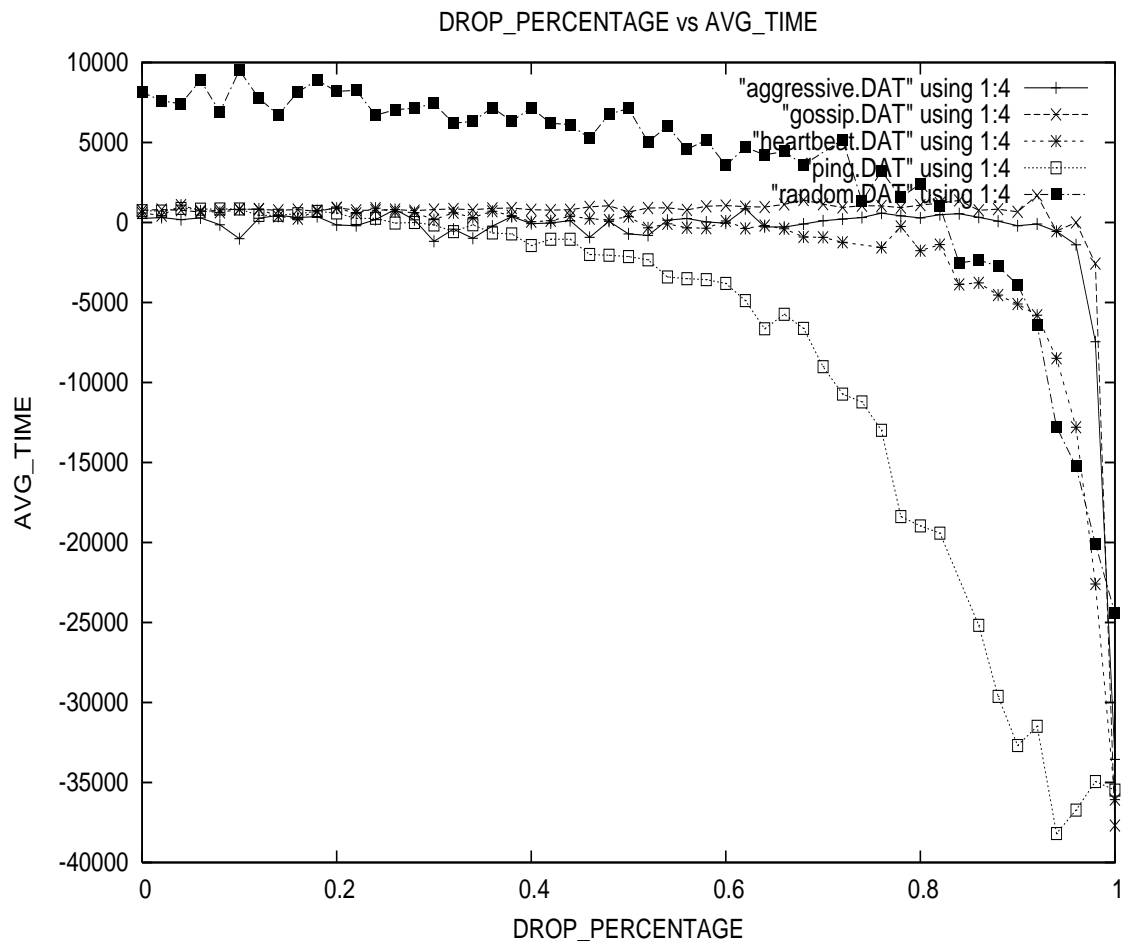


Figure 31: Network Delivery Percentage vs. Average Detection Time

Figure 31 shows the average time needed by a node in the group to detect a node failure as the number of nodes in the group grows. This graph shows the high average detection time for the Random Ping procedure, which supports Hypothesis 2. The graph also shows that as the network reliability decreases, the protocols start to detect nodes before they actually fail. This explains the number of errors seen in figure 30. This graph shows that to avoid premature and false detections the protocols would need to be dynamic so they can reconfigure themselves in response to network conditions, otherwise they would need to be reconfigured by an upper application layer.

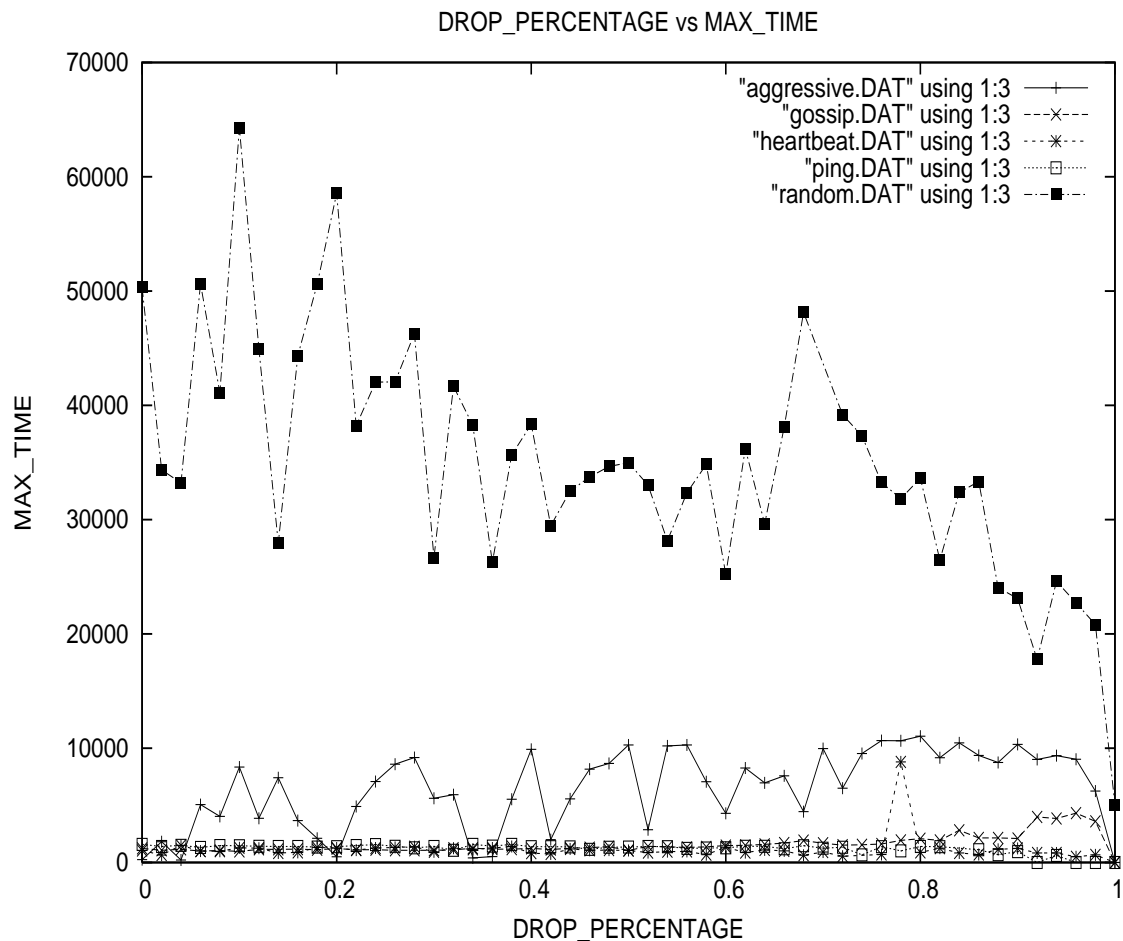


Figure 32: Network Delivery Percentage vs. Maximum Detection Time

Figure 32 is similar to figure 31 with the maximum detection time plotted instead of the average detection time. This graph proves Hypothesis 2, showing how the randomness of the ping procedure causes high detection times for the Random Ping protocol. In terms of maximum detection times the graph disproves Hypothesis 3, and instead shows that the maximum detection time of the Aggressive protocol remains relatively constant as the network reliability decreases.

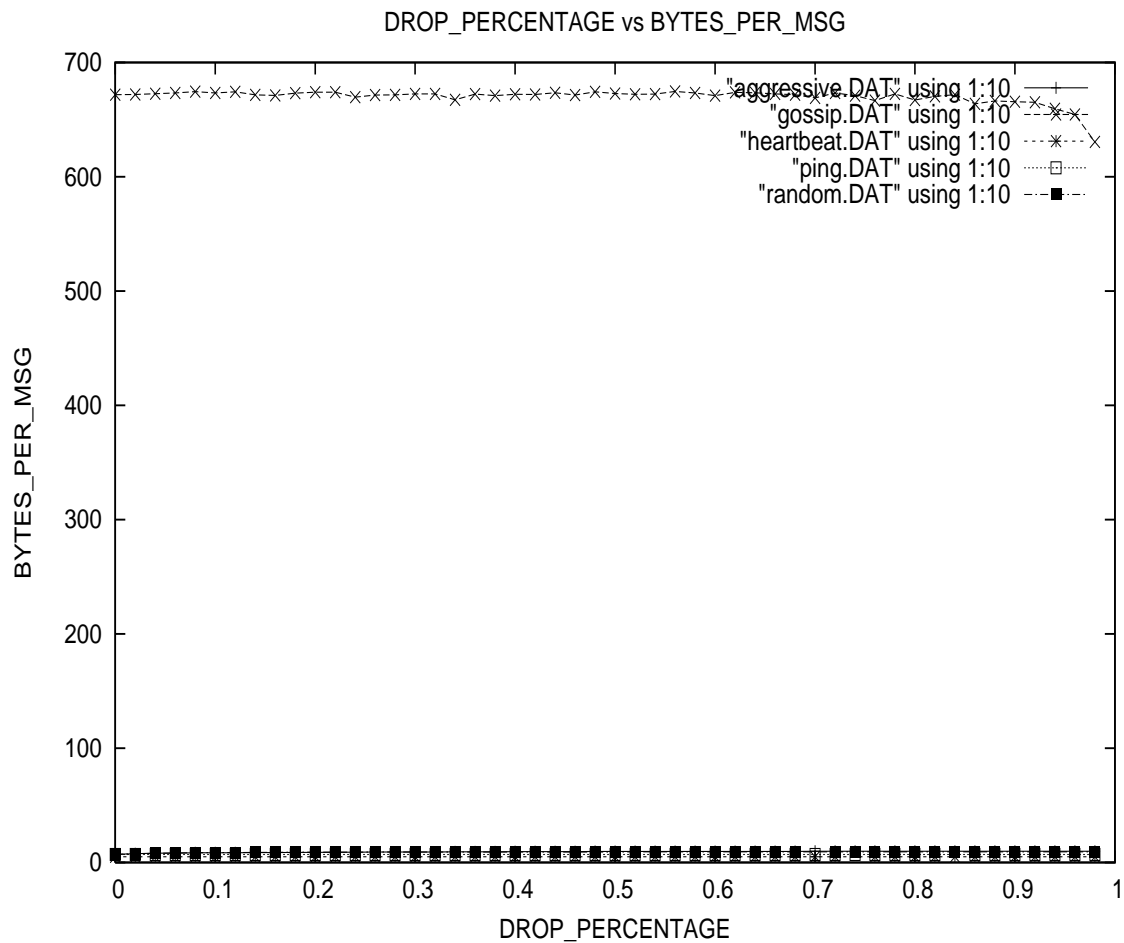


Figure 33: Network Delivery Percentage vs. Bytes Sent Per Message

Figure 33 is included to prove Hypothesis 5. It shows the size in bytes of the messages sent by the protocols as the network reliability decreases. As expected this value remains constant for each of the protocols, since the group size did not change. The graph shows how the size of the messages sent by the Gossip protocol is much larger than the size of the messages sent by the other four protocols.

## 8.4 Node Device Capabilities

The original proposed additions for Chandra, Gupta, and Goldszmidt's protocol would provide capabilities to detect weak nodes and make protocol adjustments based on those ratings. This was the reason behind the test environment's capabilities to specify the device characteristics for a node. Although these additions were abandoned, a test was still conducted to see how a protocol performs when nodes run on devices with unequal performance capabilities. As in the previous two tests the protocol parameters were held constant with the values found in the first test. The percentage of packets delivered by the network was held constant at 100%. The number of nodes in the group was held constant at 100. For each run, 50 nodes were of the default type, and the other 50 were one of the device types defined below. The data used to generate the graphs in this section is presented in appendix D. The node devices were varied from completely reliable to poor. The device type declarations used are given below.

```
TYPE = Default
PACKET_PROCESSED_PROBABILITY = 1.0
PROCESSING_DELAY_TIME_MIN = 0
PROCESSING_DELAY_TIME_MAX = 0
BUSY_SLEEP_PROBABILITY = 0.0
SLEEP_TIME_MIN = 0
SLEEP_TIME_MAX = 0
PACKET_SENT_PROBABILITY = 1.0
END_TYPE
```

```
TYPE = Slow EXTENDS Default
BUSY_SLEEP_PROBABILITY = 0.05
SLEEP_TIME_MIN = 10
SLEEP_TIME_MAX = 50
END_TYPE
```

```
TYPE = XtraSlow EXTENDS Slow
BUSY_SLEEP_PROBABILITY = 0.1
SLEEP_TIME_MIN = 30
SLEEP_TIME_MAX = 80
END_TYPE
```

```
TYPE = Lossy EXTENDS Default
PACKET_PROCESSED_PROBABILITY = 0.95
PACKET_SENT_PROBABILITY = 0.95
END_TYPE
```

```
TYPE = XtraLossy EXTENDS Lossy
PACKET_PROCESSED_PROBABILITY = 0.9
PACKET_SENT_PROBABILITY = 0.9
END_TYPE
```

```
TYPE = SlowLossy EXTENDS Slow
PACKET_PROCESSED_PROBABILITY = 0.95
PACKET_SENT_PROBABILITY = 0.95
END_TYPE
```

```
TYPE = XtraSlowXtraLossy EXTENDS Default
PACKET_PROCESSED_PROBABILITY = 0.9
BUSY_SLEEP_PROBABILITY = 0.1
SLEEP_TIME_MIN = 30
SLEEP_TIME_MAX = 80
PACKET_SENT_PROBABILITY = 0.9
END_TYPE
```

```
TYPE = Poor EXTENDS Default
PACKET_PROCESSED_PROBABILITY = 0.8
BUSY_SLEEP_PROBABILITY = 0.2
SLEEP_TIME_MIN = 60
SLEEP_TIME_MAX = 120
PACKET_SENT_PROBABILITY = 0.8
END_TYPE
```

Each device is mapped to an integer in the graphs that follow:

- 0 = Default
- 1 = Lossy
- 2 = Slow
- 3 = SlowLossy
- 4 = XtraLossy
- 5 = XtraSlow
- 6 = XtraSlowXtraLossy
- 7 = Worst

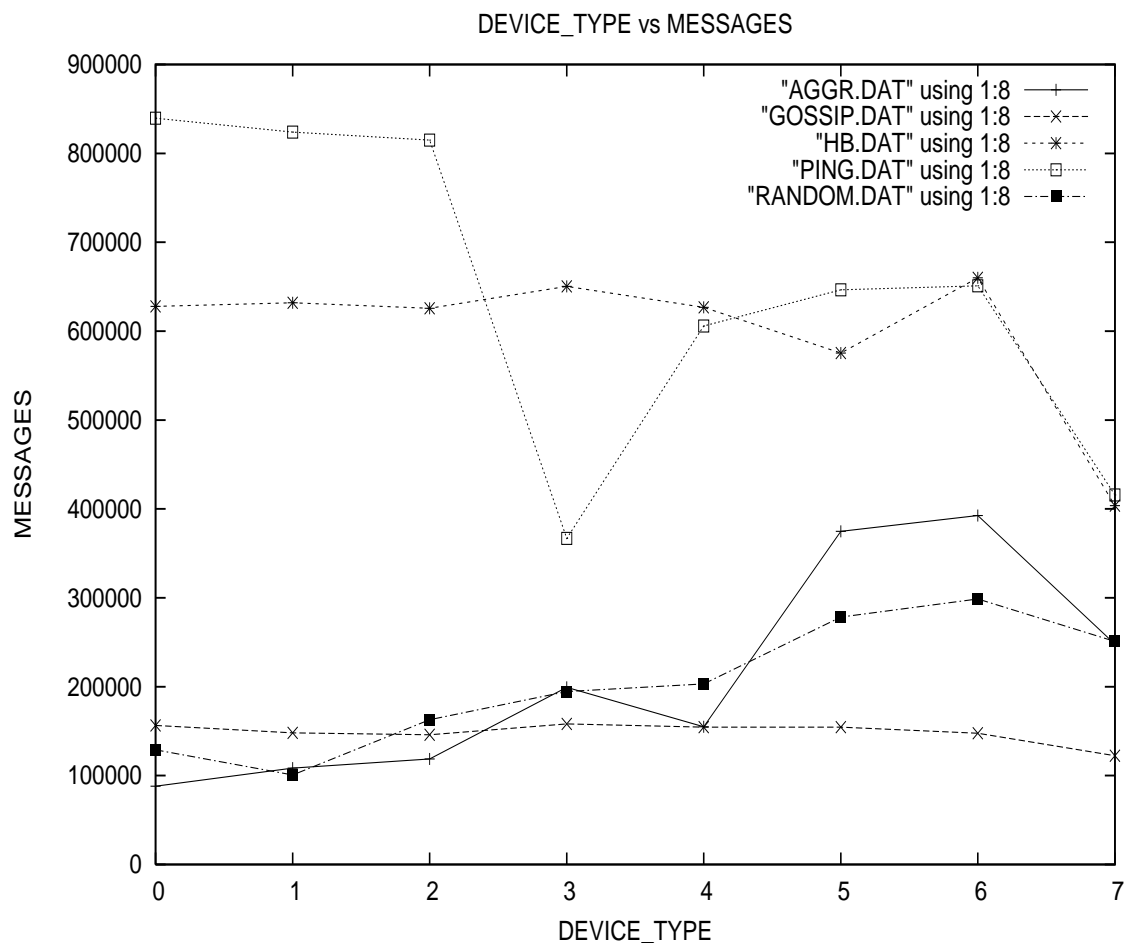


Figure 34: Device Types vs. Messages Sent

Figure 34 shows the total number of messages sent for each of the device types tested. This graph shows that the Gossip protocol's performance in terms of the number of messages sent is not affected by the device types of the nodes in the group. The graph also shows that the Aggressive and Random Ping protocols sent more messages for the tests with the less reliable devices than for the more reliable devices. This is caused by the swarm procedure executing more often due to the less reliable devices, which is expected.

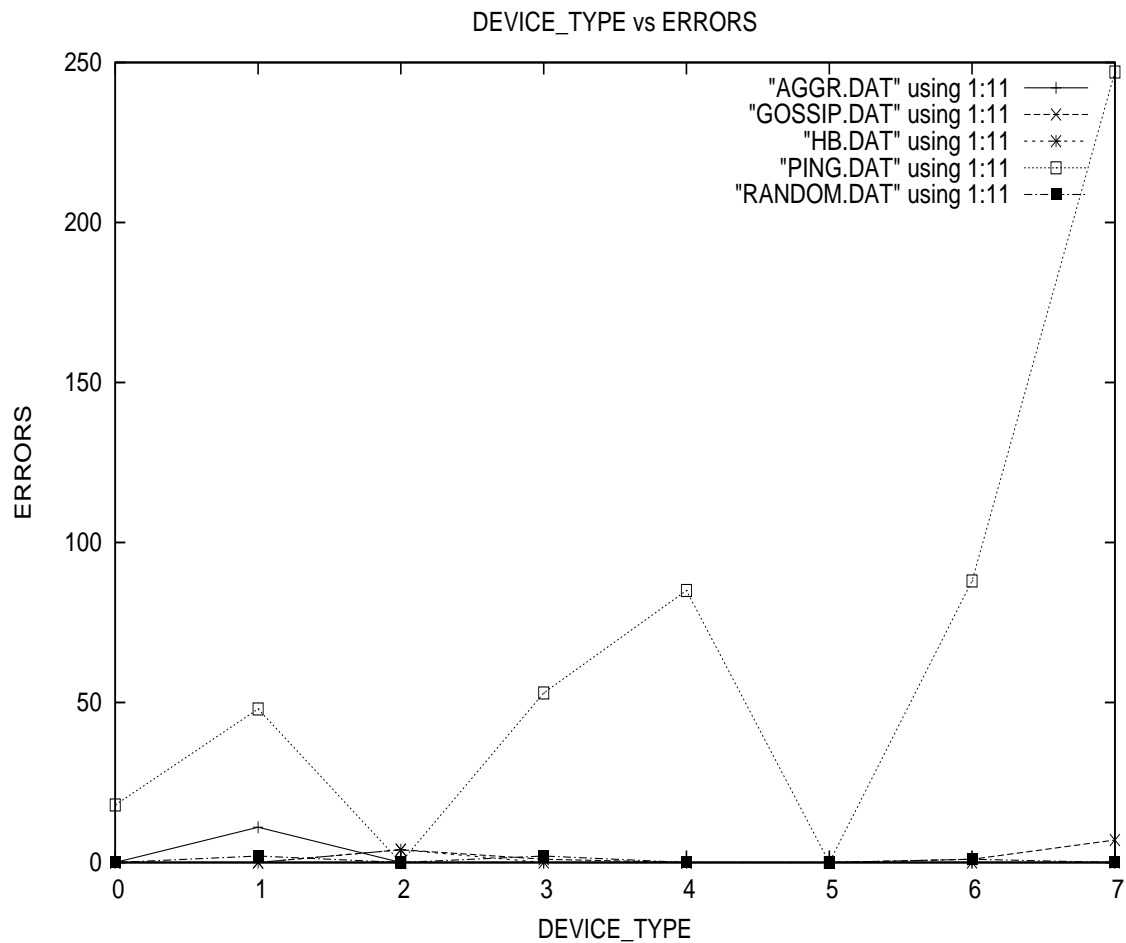


Figure 35: Device Types vs. Detection Errors

Figure 35 shows the total false number of detections for each of the device types tested. It would appear from this graph that the Ping protocol is the only protocol that has decreases in accuracy with less reliable devices. This is not completely conclusive since there are a number of uncontrolled variables in the test environment that may have caused the fluctuations. The graph does show that the other four protocols remained accurate in the face of less reliable devices.



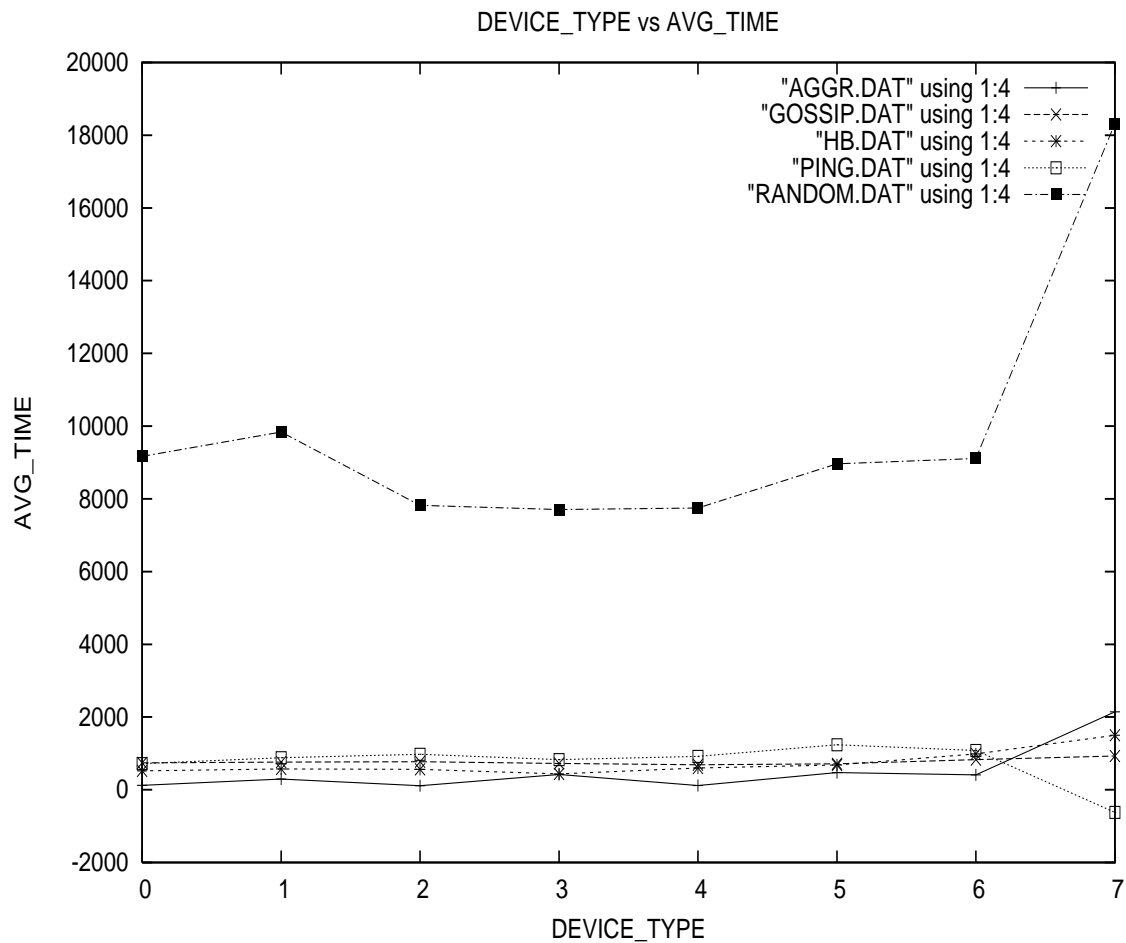


Figure 36: Device Types vs. Average Detection Time

Figure 36 shows the average time needed by a node to detect a failure for each of the device types tested. As seen in the previous two tests the Random Ping procedure has high average detection times compared to the other protocols, thus supporting Hypothesis 2. The other four protocols maintain a relatively stable average detection time for all of the device types.

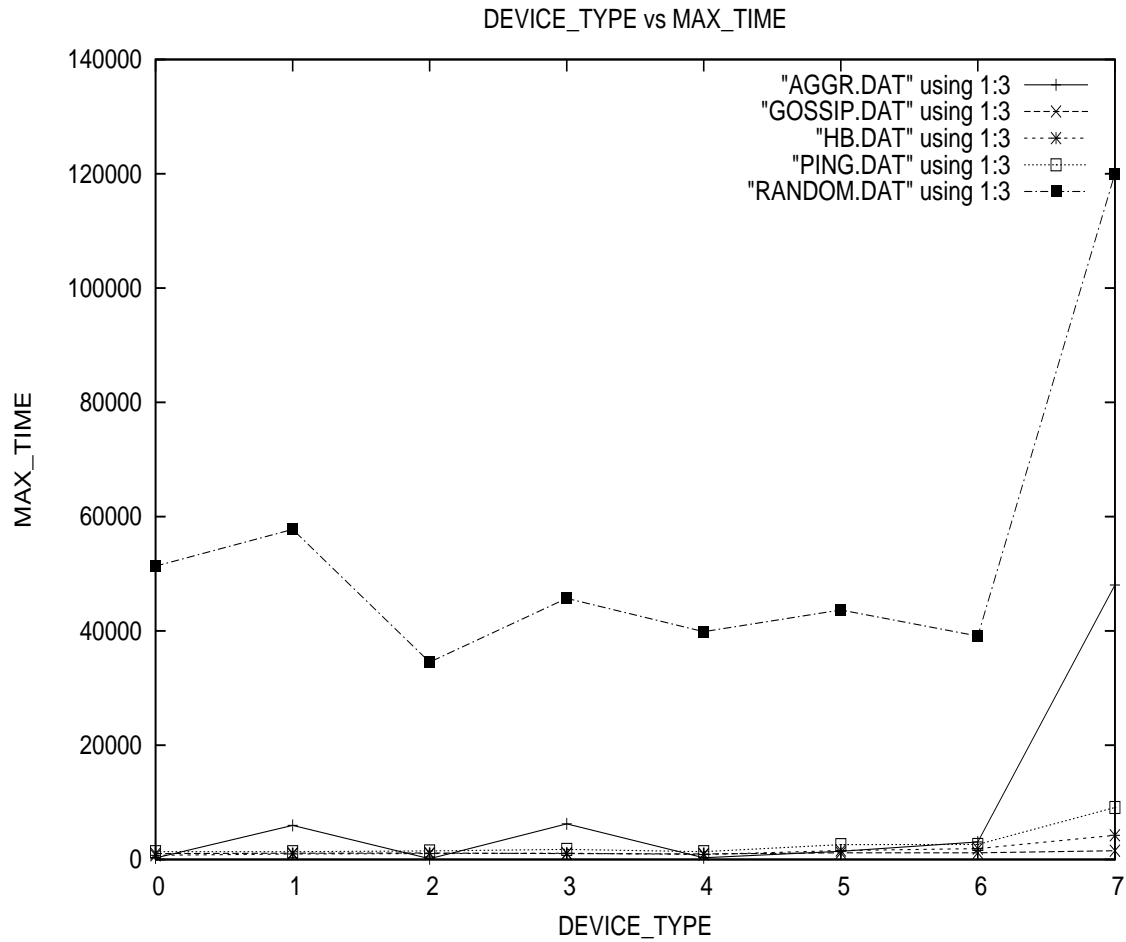


Figure 37: Device Types vs. Maximum Detection Time

Figure 37 is another graph showing the high maximum detection times for the Random Ping protocol caused by its ping selection procedure. The graph also shows that the other four protocols are not impacted by the device types in terms of maximum detection times. The Aggressive protocol showed the same patten of detection times, just on a lower scale, proving that the modification did in fact decrease the Random Ping protocol's high maximum detection time.

## 9 Conclusions

After examining the protocols in detail and viewing the test results it would appear that the Gossip protocol is the best overall protocol. As predicted in Hypothesis 5, the only obvious weakness of the Gossip protocol is the number of bytes used per gossip message. For group sizes where message fragmentation is not a problem, the size of the gossip messages should not have any negative effects on the protocol's performance.

In fairly reliable network settings the Aggressive protocol does in fact improve the high average and maximum detection times of the Random Ping protocol. The modifications implemented in the Aggressive protocol did not improve the accuracy of the Random Ping protocol. In fact they actually made the protocol less accurate. It is not clear at this time what the root cause is for the decrease in accuracy. The burst procedure could be one part of the increase in errors since it spreads a false detection to all nodes in the group. Uncontrolled variables in the test environment may have also caused some of the problems. It is also possible that the set of parameters found in the first test were not optimal.

After working on this thesis it has become apparent that knowledge sharing among the nodes in the group is the key to a fast and accurate failure detection protocol. This idea is at the heart of the Gossip protocol, and as a result it is the best among the protocols examined in this investigation. The burst procedure is an attempt to add knowledge sharing to the Random Ping protocol, but it is flawed in that it forces a detection onto other nodes in the group, and in some cases the detection may be incorrect. In a reliable network with small groups where the number of messages is not a problem, the Heartbeat and Ping protocols perform as well as the more complex protocols. If the application relying on the failure detector is not expecting large group sizes, then the Heartbeat and Ping protocols may be a good choice since they would be easy to maintain. Overall the Aggressive protocol is a start in the right direction for improving the Random Ping protocol, but it requires more work before it can be said that it is an improvement in all areas of failure detector performance.

## 10 Future Work

The work done in this thesis is far from complete, leaving many areas open for future work. One major area for future work is to design a completely simulated framework to test the protocols in. Another area for future work is to implement the protocols so that they can dynamically reconfigure themselves in response to environment changes.

There were several uncontrolled variables in the test framework used in this thesis, including network packet loss, packet transmission delay, and workstation load. These uncontrolled variables may have had an impact on the test results presented. Conducting the same tests in a simulated environment would provide more accurate and conclusive results. These results could then be used to make more accurate comparisons of the protocols, and either verify or disprove the conclusions made in this thesis. One of the main components of the simulated environment would be a class for simulating a network. This class would provide simulation variables for the percentage of packets delivered as well as the delivery delay. The network class could then be integrated with the classes implemented in this thesis.

The test results presented in this thesis show that the protocols' performance degrades as the environment changes that it has been configured for. One interesting area for future work would be to design the protocol implementations so that they can reconfigure their own parameters in response to environment changes such as when the group size grows or the network reliability decreases unexpectedly. This would require investigation into how to coordinate such a group wide parameter change as well as how to set the thresholds for the mechanisms that respond to environment changes.

## A Protocol Parameter Data

min = The minimum time for a node to detect the failure.

max = The maximum time for a node to detect the failure.

avg = The average time for a node to detect the failure.

msgs = The total number of messages sent by the nodes.

bytes = The total number of bytes sent by the nodes.

bytes/msg = The average number of bytes per message sent.

errors = The total number of false detections.

### A.1 Heartbeat Parameter Data

HB\_SD = The time interval between heartbeat messages.

HB\_TO = The Heartbeat timeout value.

HB\_CD = The time interval between scans for failed nodes.

| min  | max  | avg  | msgs    | bytes   | bytes/msg | errors | HB_SD | HB_TO | HB_CD |
|------|------|------|---------|---------|-----------|--------|-------|-------|-------|
| 350  | 1181 | 582  | 669300  | 3303300 | 4.935     | 0      | 650   | 700   | 700   |
| 183  | 917  | 707  | 665100  | 3282600 | 4.935     | 0      | 600   | 600   | 700   |
| 477  | 1069 | 733  | 637038  | 3141490 | 4.931     | 0      | 650   | 700   | 600   |
| 514  | 1087 | 747  | 748600  | 3690700 | 4.93      | 0      | 550   | 600   | 600   |
| 488  | 1329 | 766  | 690960  | 3410200 | 4.935     | 0      | 600   | 700   | 600   |
| 573  | 1289 | 787  | 466225  | 2298925 | 4.931     | 0      | 900   | 800   | 750   |
| 435  | 1430 | 924  | 546213  | 2693052 | 4.93      | 0      | 800   | 700   | 800   |
| 591  | 2205 | 955  | 605900  | 2986900 | 4.93      | 0      | 700   | 800   | 700   |
| 40   | 1409 | 1061 | 502318  | 2476790 | 4.931     | 0      | 850   | 800   | 900   |
| 1060 | 1745 | 1444 | 575000  | 2834600 | 4.93      | 0      | 750   | 800   | 900   |
| 54   | 701  | 368  | 628847  | 3099635 | 4.929     | 1      | 650   | 600   | 650   |
| 88   | 724  | 469  | 690200  | 3403100 | 4.931     | 1      | 600   | 600   | 650   |
| 167  | 716  | 484  | 852060  | 4201800 | 4.931     | 1      | 500   | 600   | 550   |
| 92   | 1595 | 844  | 988453  | 4868665 | 4.926     | 1      | 400   | 500   | 600   |
| 691  | 1354 | 939  | 686000  | 3382000 | 4.93      | 1      | 650   | 700   | 650   |
| -38  | 652  | 409  | 526825  | 2597425 | 4.93      | 2      | 800   | 700   | 700   |
| 235  | 1042 | 656  | 702100  | 3461500 | 4.93      | 2      | 600   | 700   | 700   |
| 517  | 1057 | 828  | 586604  | 2891820 | 4.93      | 2      | 700   | 700   | 600   |
| 247  | 1084 | 518  | 1107330 | 5455950 | 4.927     | 3      | 400   | 400   | 450   |
| 377  | 1228 | 667  | 878524  | 4332520 | 4.932     | 3      | 500   | 500   | 600   |
| 400  | 981  | 675  | 811500  | 4000800 | 4.93      | 3      | 500   | 500   | 550   |
| 540  | 1179 | 971  | 571978  | 2819490 | 4.929     | 3      | 750   | 700   | 650   |
| 150  | 1033 | 368  | 996700  | 4910900 | 4.927     | 5      | 400   | 500   | 500   |

|      |      |      |         |         |       |     |     |     |     |
|------|------|------|---------|---------|-------|-----|-----|-----|-----|
| 706  | 1162 | 953  | 565670  | 2788250 | 4.929 | 5   | 700 | 700 | 800 |
| 336  | 1087 | 646  | 1109884 | 5467233 | 4.926 | 6   | 400 | 500 | 450 |
| 9    | 1224 | 631  | 1153017 | 5682485 | 4.928 | 7   | 350 | 400 | 500 |
| 289  | 1284 | 565  | 926600  | 4564000 | 4.926 | 10  | 450 | 500 | 500 |
| 400  | 1255 | 790  | 731600  | 3607100 | 4.93  | 12  | 600 | 600 | 600 |
| 136  | 680  | 451  | 592081  | 2919505 | 4.931 | 13  | 700 | 600 | 600 |
| 306  | 996  | 522  | 670288  | 3305526 | 4.932 | 13  | 650 | 600 | 700 |
| 345  | 789  | 536  | 821931  | 4052231 | 4.93  | 13  | 500 | 400 | 350 |
| 535  | 1372 | 880  | 604800  | 2981500 | 4.93  | 16  | 700 | 700 | 700 |
| 281  | 1099 | 729  | 541800  | 2673400 | 4.934 | 17  | 750 | 700 | 800 |
| 411  | 913  | 707  | 657968  | 3244540 | 4.931 | 19  | 650 | 600 | 550 |
| -152 | 829  | 344  | 1032383 | 5088715 | 4.929 | 21  | 400 | 400 | 350 |
| 178  | 535  | 368  | 1194300 | 5884600 | 4.927 | 21  | 350 | 400 | 350 |
| 356  | 1312 | 608  | 678500  | 3344300 | 4.929 | 21  | 600 | 500 | 450 |
| 325  | 1146 | 761  | 791700  | 3903200 | 4.93  | 21  | 550 | 600 | 700 |
| 430  | 1393 | 923  | 519200  | 2559800 | 4.93  | 22  | 800 | 800 | 700 |
| 67   | 553  | 353  | 529100  | 2608100 | 4.929 | 23  | 800 | 700 | 600 |
| -178 | 837  | 312  | 910022  | 4483810 | 4.927 | 29  | 450 | 400 | 400 |
| 295  | 670  | 485  | 818900  | 4039200 | 4.932 | 31  | 500 | 500 | 450 |
| 630  | 1432 | 1231 | 574386  | 2831530 | 4.93  | 36  | 700 | 800 | 900 |
| 44   | 953  | 273  | 1034529 | 5098945 | 4.929 | 40  | 400 | 400 | 300 |
| 112  | 669  | 467  | 734300  | 3619700 | 4.929 | 40  | 600 | 600 | 500 |
| 142  | 636  | 392  | 462246  | 2281630 | 4.936 | 41  | 900 | 800 | 850 |
| 87   | 519  | 304  | 836089  | 4124045 | 4.933 | 46  | 500 | 400 | 450 |
| 35   | 885  | 366  | 845153  | 4168265 | 4.932 | 53  | 500 | 400 | 500 |
| 229  | 906  | 391  | 679995  | 3352375 | 4.93  | 58  | 650 | 600 | 500 |
| 501  | 1205 | 865  | 980236  | 4831280 | 4.929 | 58  | 450 | 500 | 550 |
| 736  | 1549 | 1291 | 532637  | 2626385 | 4.931 | 64  | 800 | 800 | 850 |
| 557  | 1455 | 870  | 686808  | 3386740 | 4.931 | 68  | 650 | 700 | 750 |
| 224  | 1269 | 577  | 1199134 | 5907231 | 4.926 | 75  | 350 | 400 | 300 |
| 204  | 1011 | 442  | 1589551 | 7843055 | 4.934 | 79  | 250 | 300 | 400 |
| 123  | 1012 | 294  | 972207  | 4795735 | 4.933 | 82  | 400 | 300 | 350 |
| 117  | 1085 | 389  | 940489  | 4638345 | 4.932 | 88  | 450 | 500 | 450 |
| -40  | 990  | 410  | 2012042 | 9917648 | 4.929 | 90  | 200 | 300 | 400 |
| 18   | 1279 | 439  | 1612483 | 7941315 | 4.925 | 95  | 250 | 300 | 300 |
| -363 | 1377 | 406  | 1362233 | 6712269 | 4.927 | 96  | 300 | 300 | 200 |
| 80   | 718  | 300  | 1271900 | 6274300 | 4.933 | 101 | 300 | 300 | 400 |

|       |      |     |         |          |       |     |     |     |     |
|-------|------|-----|---------|----------|-------|-----|-----|-----|-----|
| -10   | 1428 | 515 | 1414298 | 6974420  | 4.931 | 107 | 300 | 300 | 350 |
| 486   | 1232 | 863 | 682031  | 3361079  | 4.928 | 118 | 600 | 500 | 600 |
| 129   | 1943 | 634 | 1363377 | 6716283  | 4.926 | 121 | 300 | 400 | 500 |
| 39    | 1199 | 312 | 1659831 | 8176641  | 4.926 | 127 | 250 | 300 | 200 |
| -121  | 525  | 302 | 1409805 | 6946525  | 4.927 | 142 | 300 | 200 | 250 |
| -325  | 913  | 263 | 1217655 | 6004875  | 4.932 | 148 | 300 | 200 | 100 |
| -170  | 1266 | 222 | 1472270 | 7252250  | 4.926 | 162 | 250 | 200 | 200 |
| -604  | 712  | 280 | 1531107 | 7547035  | 4.929 | 206 | 250 | 200 | 300 |
| -277  | 529  | 99  | 1323016 | 6520397  | 4.928 | 210 | 300 | 200 | 200 |
| -354  | 1060 | 283 | 1457092 | 7182500  | 4.929 | 262 | 300 | 200 | 150 |
| 94    | 864  | 441 | 602134  | 2969370  | 4.931 | 270 | 700 | 700 | 650 |
| -448  | 914  | 72  | 1975122 | 9720910  | 4.922 | 324 | 200 | 200 | 250 |
| -881  | 595  | 93  | 1808766 | 8905036  | 4.923 | 355 | 200 | 200 | 150 |
| -406  | 700  | 119 | 1272075 | 6269253  | 4.928 | 391 | 300 | 200 | 300 |
| 74    | 598  | 190 | 970000  | 4782700  | 4.931 | 397 | 400 | 300 | 200 |
| -267  | 2292 | 358 | 1989562 | 9794595  | 4.923 | 402 | 200 | 300 | 200 |
| -1107 | 1353 | 321 | 1891075 | 9312189  | 4.924 | 426 | 200 | 200 | 300 |
| -366  | 1350 | 349 | 2448914 | 12046075 | 4.919 | 448 | 150 | 200 | 250 |
| -167  | 1332 | 220 | 1751553 | 8621320  | 4.922 | 468 | 200 | 200 | 100 |
| -988  | 1413 | 349 | 2291517 | 11279011 | 4.922 | 489 | 150 | 200 | 200 |
| -683  | 1344 | 377 | 3159074 | 15552401 | 4.923 | 490 | 100 | 200 | 100 |
| -1449 | 1551 | 191 | 2844318 | 13989397 | 4.918 | 500 | 100 | 200 | 150 |
| -1443 | 1247 | 294 | 2877595 | 14146445 | 4.916 | 560 | 100 | 200 | 250 |

## A.2 Ping Parameter Data

PING\_SD = The time interval between ping messages.

ACK\_TO = The timeout value for a ping acknowledgment message.

| min  | max  | avg | msgs    | bytes   | bytes/msg | errors | PING_SD | ACK_TO |
|------|------|-----|---------|---------|-----------|--------|---------|--------|
| -237 | 2351 | 815 | 1124871 | 7794153 | 6.929     | 0      | 750     | 400    |
| 456  | 1526 | 838 | 950688  | 6588423 | 6.93      | 0      | 900     | 300    |
| 402  | 2052 | 852 | 859875  | 5958708 | 6.93      | 0      | 1000    | 400    |
| 520  | 2483 | 884 | 1132003 | 7843583 | 6.929     | 0      | 700     | 500    |
| 512  | 1492 | 888 | 1030814 | 7143489 | 6.93      | 0      | 850     | 500    |
| 508  | 1542 | 895 | 1348745 | 9351236 | 6.933     | 0      | 650     | 500    |
| 312  | 1362 | 960 | 834365  | 5782778 | 6.931     | 0      | 1000    | 300    |
| 413  | 2483 | 962 | 883085  | 6119024 | 6.929     | 0      | 950     | 400    |

|       |      |      |         |          |       |     |      |      |
|-------|------|------|---------|----------|-------|-----|------|------|
| 501   | 1962 | 988  | 1107396 | 7673035  | 6.929 | 0   | 750  | 500  |
| 530   | 1550 | 989  | 775454  | 5374329  | 6.931 | 0   | 1050 | 500  |
| 372   | 1500 | 1024 | 821425  | 5692440  | 6.93  | 0   | 1100 | 600  |
| 512   | 2025 | 1044 | 1060008 | 7345666  | 6.93  | 0   | 800  | 500  |
| 307   | 1619 | 1099 | 782350  | 5421570  | 6.93  | 0   | 1100 | 300  |
| 649   | 2275 | 1170 | 1012591 | 7014017  | 6.927 | 0   | 850  | 600  |
| 708   | 2094 | 1201 | 990610  | 6861421  | 6.926 | 0   | 800  | 700  |
| 808   | 2362 | 1235 | 896631  | 6213442  | 6.93  | 0   | 1000 | 800  |
| 804   | 1643 | 1257 | 1039821 | 7204567  | 6.929 | 0   | 800  | 800  |
| 893   | 2041 | 1265 | 825737  | 5722420  | 6.93  | 0   | 1100 | 700  |
| 670   | 1861 | 1295 | 999831  | 6924967  | 6.926 | 0   | 850  | 700  |
| 729   | 2011 | 1329 | 848094  | 5877375  | 6.93  | 0   | 1050 | 700  |
| 993   | 1974 | 1343 | 885223  | 6135130  | 6.931 | 0   | 1000 | 1000 |
| 890   | 1789 | 1376 | 1017996 | 7053999  | 6.929 | 0   | 900  | 900  |
| 800   | 1842 | 1387 | 833356  | 5774916  | 6.93  | 0   | 1050 | 800  |
| 949   | 2378 | 1398 | 790088  | 5474951  | 6.93  | 0   | 1100 | 900  |
| 758   | 1970 | 1442 | 875488  | 6066473  | 6.929 | 0   | 950  | 700  |
| 874   | 1900 | 1442 | 742039  | 5142283  | 6.93  | 0   | 1100 | 800  |
| 909   | 3032 | 1518 | 806156  | 5586369  | 6.93  | 0   | 1050 | 900  |
| 856   | 2742 | 1527 | 863718  | 5984321  | 6.929 | 0   | 950  | 900  |
| 1058  | 2696 | 1545 | 779994  | 5404679  | 6.929 | 0   | 1100 | 1000 |
| 925   | 2716 | 1591 | 876849  | 6076453  | 6.93  | 0   | 1000 | 900  |
| 1070  | 2833 | 1695 | 823089  | 5703409  | 6.929 | 0   | 1050 | 1000 |
| 697   | 1965 | 996  | 1177690 | 8159939  | 6.929 | 1   | 700  | 700  |
| 739   | 3050 | 1124 | 910641  | 6308153  | 6.927 | 1   | 900  | 700  |
| 882   | 2675 | 1315 | 952302  | 6595315  | 6.926 | 1   | 850  | 800  |
| 657   | 1667 | 1322 | 887053  | 6147617  | 6.93  | 2   | 1000 | 600  |
| 309   | 1776 | 816  | 1187843 | 8231339  | 6.93  | 9   | 700  | 300  |
| 566   | 1512 | 1128 | 946800  | 6561193  | 6.93  | 9   | 950  | 600  |
| 365   | 1600 | 836  | 1008352 | 6987588  | 6.93  | 11  | 800  | 300  |
| 749   | 2140 | 1163 | 891263  | 6174178  | 6.927 | 20  | 900  | 600  |
| 611   | 2093 | 1066 | 1114246 | 7718537  | 6.927 | 35  | 750  | 600  |
| -3617 | 6238 | 798  | 1489948 | 10317131 | 6.924 | 47  | 500  | 400  |
| -77   | 1382 | 753  | 1404361 | 9730756  | 6.929 | 95  | 600  | 500  |
| -253  | 1311 | 544  | 1452724 | 10072159 | 6.933 | 120 | 600  | 300  |
| -10   | 1988 | 949  | 1474792 | 10217668 | 6.928 | 133 | 600  | 600  |
| -441  | 844  | 277  | 1934249 | 13401935 | 6.929 | 223 | 400  | 400  |



|      |      |     |         |          |       |     |     |     |
|------|------|-----|---------|----------|-------|-----|-----|-----|
| -104 | 1316 | 276 | 1748466 | 12115234 | 6.929 | 309 | 450 | 400 |
| -261 | 1481 | 511 | 1652224 | 11446451 | 6.928 | 312 | 400 | 300 |

### A.3 Gossip Parameter Data

SD = The time interval between gossip messages.

TO = The timeout value for the Gossip protocol.

B = The number of nodes that each gossip message is sent to.

CD = The time interval between scans for failed nodes.

| min | max  | avg | msgs   | bytes    | bytes/msg | errors | SD  | TO  | B | CD  |
|-----|------|-----|--------|----------|-----------|--------|-----|-----|---|-----|
| 412 | 1033 | 602 | 162830 | 1.1e+08  | 674.348   | 0      | 100 | 450 | 4 | 450 |
| 419 | 980  | 676 | 200098 | 1.35e+08 | 674.077   | 0      | 100 | 400 | 5 | 400 |
| 487 | 993  | 768 | 182485 | 1.23e+08 | 671.394   | 0      | 100 | 400 | 5 | 400 |
| 432 | 1153 | 774 | 157452 | 1.06e+08 | 673.545   | 0      | 100 | 450 | 4 | 450 |
| 149 | 595  | 363 | 176410 | 1.18e+08 | 670.36    | 1      | 100 | 200 | 5 | 250 |
| 190 | 829  | 410 | 218095 | 1.46e+08 | 671.308   | 1      | 100 | 250 | 6 | 300 |
| 220 | 1248 | 413 | 181540 | 1.22e+08 | 671.232   | 1      | 100 | 200 | 5 | 250 |
| 203 | 665  | 415 | 167853 | 1.12e+08 | 668.726   | 1      | 125 | 200 | 5 | 250 |
| 268 | 822  | 523 | 151230 | 99485316 | 657.841   | 1      | 150 | 250 | 6 | 300 |
| 29  | 940  | 525 | 95775  | 61655475 | 643.753   | 1      | 200 | 300 | 5 | 375 |
| 486 | 1114 | 727 | 148660 | 99843352 | 671.622   | 1      | 100 | 400 | 4 | 400 |
| 485 | 1175 | 824 | 149248 | 1e+08    | 671.943   | 1      | 100 | 450 | 4 | 450 |
| 116 | 429  | 258 | 188885 | 1.27e+08 | 672.318   | 2      | 100 | 200 | 5 | 125 |
| 235 | 645  | 429 | 160228 | 1.08e+08 | 673.948   | 2      | 100 | 200 | 4 | 200 |
| 274 | 769  | 582 | 101215 | 65478380 | 646.924   | 2      | 200 | 300 | 5 | 350 |
| 380 | 978  | 665 | 148680 | 99885428 | 671.815   | 2      | 100 | 450 | 4 | 450 |
| 116 | 943  | 500 | 93950  | 60380580 | 642.688   | 3      | 225 | 300 | 5 | 400 |
| 358 | 904  | 640 | 92336  | 60341188 | 653.496   | 3      | 150 | 350 | 4 | 250 |
| 216 | 574  | 402 | 185610 | 1.25e+08 | 671.897   | 4      | 100 | 200 | 5 | 275 |
| 285 | 764  | 494 | 155064 | 1.04e+08 | 673.073   | 4      | 100 | 250 | 4 | 300 |
| 160 | 582  | 387 | 124680 | 81944535 | 657.239   | 5      | 150 | 250 | 5 | 300 |
| 57  | 598  | 388 | 150108 | 1.01e+08 | 672.104   | 5      | 100 | 200 | 4 | 300 |
| 301 | 720  | 522 | 162072 | 1.09e+08 | 674.336   | 5      | 100 | 300 | 4 | 250 |
| 405 | 847  | 614 | 156956 | 1.06e+08 | 673.418   | 6      | 100 | 400 | 4 | 250 |
| 103 | 664  | 391 | 82065  | 52036155 | 634.085   | 7      | 250 | 300 | 5 | 375 |
| 139 | 519  | 302 | 170480 | 1.14e+08 | 669.257   | 8      | 125 | 200 | 5 | 300 |
| 20  | 538  | 320 | 166390 | 1.11e+08 | 668.457   | 11     | 125 | 200 | 5 | 225 |
| 191 | 661  | 358 | 185470 | 1.25e+08 | 671.871   | 11     | 100 | 200 | 5 | 250 |

|      |      |     |        |          |         |    |     |     |   |     |
|------|------|-----|--------|----------|---------|----|-----|-----|---|-----|
| 185  | 631  | 365 | 206694 | 1.38e+08 | 669.565 | 11 | 100 | 200 | 6 | 250 |
| 205  | 592  | 364 | 228498 | 1.54e+08 | 672.529 | 12 | 100 | 200 | 6 | 300 |
| 96   | 664  | 419 | 147568 | 99109748 | 671.621 | 13 | 100 | 250 | 4 | 250 |
| 266  | 928  | 541 | 187835 | 1.26e+08 | 672.263 | 13 | 100 | 250 | 5 | 300 |
| 218  | 627  | 416 | 150132 | 1.01e+08 | 672.101 | 14 | 100 | 200 | 4 | 250 |
| 288  | 801  | 554 | 103030 | 66750670 | 647.876 | 14 | 200 | 300 | 5 | 325 |
| 134  | 565  | 277 | 189000 | 1.27e+08 | 672.42  | 16 | 100 | 200 | 5 | 175 |
| 162  | 519  | 309 | 206181 | 1.39e+08 | 674.94  | 16 | 100 | 200 | 5 | 150 |
| -88  | 773  | 441 | 88625  | 56636960 | 639.063 | 17 | 225 | 300 | 5 | 300 |
| 157  | 638  | 319 | 232766 | 1.57e+08 | 673.244 | 22 | 100 | 200 | 6 | 200 |
| 46   | 339  | 193 | 189574 | 1.25e+08 | 660.969 | 24 | 150 | 200 | 7 | 150 |
| -27  | 808  | 373 | 98108  | 64393596 | 656.354 | 25 | 150 | 250 | 4 | 300 |
| 114  | 558  | 320 | 148650 | 98770175 | 664.448 | 27 | 125 | 200 | 5 | 275 |
| 188  | 672  | 426 | 146500 | 98352976 | 671.351 | 28 | 100 | 200 | 4 | 250 |
| -11  | 702  | 443 | 93315  | 59933755 | 642.274 | 28 | 225 | 300 | 5 | 250 |
| 28   | 805  | 453 | 134920 | 89132675 | 660.634 | 28 | 150 | 200 | 5 | 300 |
| 180  | 607  | 402 | 186820 | 1.26e+08 | 672.078 | 29 | 100 | 200 | 5 | 300 |
| 37   | 767  | 476 | 95800  | 61671630 | 643.754 | 29 | 200 | 300 | 5 | 225 |
| 332  | 1033 | 576 | 99770  | 64464285 | 646.129 | 29 | 225 | 300 | 5 | 350 |
| 61   | 642  | 358 | 189290 | 1.27e+08 | 672.386 | 32 | 100 | 200 | 5 | 200 |
| -20  | 950  | 536 | 105806 | 68700972 | 649.311 | 32 | 200 | 300 | 5 | 400 |
| 12   | 443  | 279 | 159954 | 1.06e+08 | 660.244 | 36 | 150 | 200 | 6 | 150 |
| 35   | 694  | 419 | 119060 | 77999200 | 655.125 | 38 | 150 | 200 | 5 | 275 |
| 186  | 724  | 443 | 91600  | 58727210 | 641.127 | 38 | 225 | 300 | 5 | 325 |
| 56   | 768  | 477 | 92360  | 59259135 | 641.61  | 39 | 225 | 300 | 5 | 200 |
| 176  | 735  | 399 | 91495  | 58650370 | 641.023 | 41 | 225 | 300 | 5 | 275 |
| 255  | 649  | 446 | 103235 | 66897200 | 648.009 | 41 | 200 | 300 | 5 | 200 |
| 194  | 788  | 397 | 152704 | 1.03e+08 | 672.629 | 43 | 100 | 200 | 4 | 250 |
| -70  | 788  | 397 | 81540  | 51669740 | 633.674 | 44 | 250 | 300 | 5 | 325 |
| -86  | 908  | 616 | 103290 | 66929330 | 647.975 | 45 | 200 | 300 | 5 | 250 |
| -10  | 948  | 359 | 126125 | 82959960 | 657.76  | 46 | 175 | 200 | 5 | 200 |
| -37  | 881  | 543 | 79135  | 49979300 | 631.57  | 46 | 250 | 300 | 5 | 200 |
| 19   | 919  | 574 | 88285  | 56396685 | 638.803 | 46 | 225 | 300 | 5 | 225 |
| -234 | 459  | 249 | 124580 | 81871570 | 657.181 | 49 | 150 | 200 | 5 | 150 |
| 165  | 659  | 371 | 168390 | 1.13e+08 | 668.881 | 50 | 125 | 200 | 5 | 150 |
| -90  | 600  | 298 | 63205  | 38801070 | 613.892 | 54 | 300 | 300 | 5 | 225 |
| -271 | 832  | 532 | 84940  | 54051470 | 636.349 | 54 | 250 | 300 | 5 | 350 |

|      |      |     |        |          |         |     |     |     |   |     |
|------|------|-----|--------|----------|---------|-----|-----|-----|---|-----|
| -13  | 684  | 414 | 109770 | 71480855 | 651.188 | 55  | 175 | 200 | 5 | 300 |
| 13   | 844  | 381 | 123045 | 80798060 | 656.655 | 56  | 150 | 200 | 5 | 175 |
| 49   | 969  | 529 | 95510  | 61468970 | 643.587 | 56  | 225 | 300 | 5 | 375 |
| 1    | 689  | 397 | 117550 | 76944210 | 654.566 | 57  | 175 | 200 | 5 | 225 |
| -51  | 622  | 300 | 117215 | 76708040 | 654.422 | 60  | 175 | 200 | 5 | 175 |
| -26  | 628  | 338 | 131475 | 86714805 | 659.554 | 60  | 150 | 200 | 5 | 250 |
| -84  | 505  | 241 | 127360 | 83824990 | 658.174 | 65  | 150 | 200 | 5 | 200 |
| -189 | 821  | 379 | 54000  | 32334025 | 598.778 | 65  | 350 | 300 | 5 | 325 |
| -46  | 943  | 547 | 71160  | 44379825 | 623.663 | 66  | 300 | 300 | 5 | 400 |
| 158  | 585  | 380 | 147820 | 98189705 | 664.252 | 67  | 125 | 200 | 5 | 200 |
| 22   | 936  | 556 | 74880  | 46986170 | 627.486 | 67  | 275 | 300 | 5 | 350 |
| -23  | 749  | 424 | 100565 | 65020700 | 646.554 | 72  | 200 | 200 | 5 | 275 |
| -69  | 764  | 436 | 65035  | 40080855 | 616.297 | 72  | 300 | 300 | 5 | 300 |
| 66   | 631  | 412 | 115915 | 75792175 | 653.86  | 73  | 175 | 200 | 5 | 250 |
| -36  | 901  | 530 | 78855  | 49777500 | 631.254 | 73  | 250 | 300 | 5 | 250 |
| 11   | 554  | 316 | 112510 | 73403595 | 652.418 | 75  | 175 | 200 | 5 | 275 |
| 144  | 630  | 347 | 102690 | 66509250 | 647.67  | 76  | 200 | 200 | 5 | 300 |
| 150  | 672  | 417 | 100845 | 65215305 | 646.689 | 76  | 200 | 200 | 5 | 250 |
| -79  | 811  | 487 | 75370  | 47335725 | 628.045 | 76  | 250 | 300 | 5 | 300 |
| 11   | 652  | 329 | 88640  | 56650880 | 639.112 | 77  | 225 | 200 | 5 | 250 |
| -53  | 1043 | 699 | 76190  | 47910625 | 628.831 | 77  | 275 | 300 | 5 | 200 |
| 86   | 817  | 551 | 78415  | 49472955 | 630.912 | 79  | 250 | 300 | 5 | 225 |
| -91  | 749  | 456 | 74975  | 47059395 | 627.668 | 82  | 275 | 300 | 5 | 375 |
| -130 | 842  | 420 | 78440  | 49489500 | 630.922 | 83  | 275 | 300 | 5 | 400 |
| 2    | 464  | 323 | 171880 | 1.15e+08 | 669.5   | 84  | 125 | 200 | 5 | 100 |
| 82   | 743  | 480 | 107780 | 70083365 | 650.245 | 87  | 200 | 300 | 5 | 300 |
| -245 | 1294 | 549 | 60370  | 36810110 | 609.742 | 87  | 325 | 300 | 5 | 400 |
| 71   | 483  | 192 | 212093 | 1.4e+08  | 660.112 | 88  | 150 | 200 | 8 | 150 |
| -258 | 726  | 420 | 89350  | 57143555 | 639.547 | 90  | 225 | 200 | 5 | 275 |
| -564 | 926  | 522 | 69805  | 43431605 | 622.185 | 93  | 300 | 300 | 5 | 275 |
| -50  | 673  | 384 | 108404 | 71621160 | 660.687 | 94  | 150 | 200 | 4 | 150 |
| -170 | 1049 | 618 | 65225  | 40214335 | 616.548 | 95  | 325 | 300 | 5 | 375 |
| -82  | 634  | 304 | 96090  | 61880825 | 643.988 | 101 | 200 | 200 | 5 | 225 |
| -37  | 1139 | 633 | 69270  | 43058540 | 621.604 | 102 | 300 | 300 | 5 | 375 |
| -54  | 728  | 458 | 86460  | 55116440 | 637.479 | 103 | 250 | 300 | 5 | 275 |
| -126 | 780  | 397 | 78105  | 49259585 | 630.684 | 105 | 275 | 300 | 5 | 275 |
| -8   | 888  | 531 | 64700  | 39843025 | 615.812 | 105 | 325 | 300 | 5 | 225 |

|      |      |     |        |          |         |     |     |     |   |     |
|------|------|-----|--------|----------|---------|-----|-----|-----|---|-----|
| -128 | 702  | 404 | 72730  | 45481190 | 625.343 | 106 | 300 | 300 | 5 | 200 |
| 137  | 917  | 519 | 82775  | 52531465 | 634.63  | 113 | 250 | 300 | 5 | 400 |
| -380 | 1053 | 536 | 56630  | 34181610 | 603.595 | 114 | 350 | 300 | 5 | 375 |
| -10  | 472  | 227 | 96025  | 61831185 | 643.907 | 118 | 200 | 200 | 5 | 200 |
| -39  | 878  | 468 | 55375  | 33304515 | 601.436 | 118 | 350 | 300 | 5 | 250 |
| -24  | 571  | 348 | 105195 | 68271695 | 649.001 | 121 | 200 | 200 | 5 | 125 |
| -213 | 767  | 346 | 74410  | 46659575 | 627.061 | 122 | 275 | 300 | 5 | 250 |
| -385 | 764  | 385 | 58955  | 35814440 | 607.488 | 123 | 375 | 300 | 5 | 275 |
| -81  | 1043 | 642 | 68065  | 42208590 | 620.122 | 123 | 325 | 300 | 5 | 350 |
| -330 | 964  | 424 | 63190  | 38785195 | 613.787 | 127 | 300 | 300 | 5 | 250 |
| 43   | 513  | 350 | 140230 | 92863505 | 662.223 | 129 | 150 | 200 | 5 | 100 |
| -151 | 686  | 343 | 91945  | 58974740 | 641.413 | 130 | 225 | 200 | 5 | 300 |
| 12   | 756  | 345 | 105000 | 68135320 | 648.908 | 136 | 200 | 200 | 5 | 150 |
| -360 | 1170 | 590 | 49105  | 28998405 | 590.539 | 136 | 400 | 300 | 5 | 350 |
| -109 | 994  | 610 | 66515  | 41122415 | 618.243 | 139 | 325 | 300 | 5 | 300 |
| -77  | 767  | 402 | 64475  | 39689505 | 615.58  | 140 | 325 | 300 | 5 | 250 |
| -184 | 844  | 436 | 71875  | 44879975 | 624.417 | 144 | 300 | 300 | 5 | 350 |
| -429 | 777  | 367 | 47270  | 27894350 | 590.107 | 146 | 400 | 300 | 5 | 400 |
| -190 | 594  | 261 | 85615  | 54528940 | 636.909 | 150 | 250 | 200 | 5 | 250 |
| -339 | 1285 | 628 | 55030  | 33060665 | 600.775 | 152 | 375 | 300 | 5 | 375 |
| -148 | 1045 | 545 | 57525  | 34812235 | 605.167 | 153 | 350 | 300 | 5 | 400 |
| -220 | 1067 | 442 | 50960  | 30237400 | 593.356 | 157 | 400 | 300 | 5 | 375 |
| -13  | 778  | 444 | 76220  | 47930895 | 628.849 | 158 | 250 | 200 | 5 | 225 |
| -171 | 831  | 443 | 60510  | 36909265 | 609.97  | 159 | 350 | 300 | 5 | 225 |
| -101 | 946  | 493 | 53150  | 31742960 | 597.233 | 160 | 375 | 300 | 5 | 300 |
| -133 | 652  | 337 | 90005  | 57610135 | 640.077 | 161 | 225 | 200 | 5 | 225 |
| 95   | 532  | 278 | 136480 | 90226235 | 661.095 | 163 | 150 | 200 | 5 | 225 |
| -91  | 676  | 431 | 82160  | 52095755 | 634.077 | 163 | 250 | 200 | 5 | 300 |
| -411 | 1077 | 571 | 59350  | 36092340 | 608.127 | 164 | 350 | 300 | 5 | 300 |
| -96  | 1026 | 531 | 51780  | 30798570 | 594.797 | 169 | 400 | 300 | 5 | 300 |
| -12  | 903  | 540 | 70435  | 43870520 | 622.851 | 169 | 275 | 300 | 5 | 325 |
| -81  | 941  | 571 | 69705  | 43361250 | 622.068 | 170 | 275 | 300 | 5 | 300 |
| -204 | 716  | 386 | 64015  | 39360200 | 614.859 | 171 | 325 | 300 | 5 | 325 |
| -152 | 976  | 531 | 55075  | 33093110 | 600.874 | 173 | 375 | 300 | 5 | 325 |
| -139 | 599  | 290 | 86905  | 55430535 | 637.829 | 175 | 225 | 200 | 5 | 200 |
| -240 | 577  | 306 | 92015  | 59018885 | 641.405 | 177 | 200 | 200 | 5 | 175 |
| -120 | 1210 | 616 | 56045  | 33771675 | 602.581 | 177 | 375 | 300 | 5 | 400 |

|      |      |     |        |          |         |     |     |     |   |     |
|------|------|-----|--------|----------|---------|-----|-----|-----|---|-----|
| -133 | 594  | 312 | 121300 | 79578530 | 656.047 | 180 | 175 | 200 | 5 | 125 |
| -127 | 1440 | 498 | 55280  | 33232205 | 601.161 | 183 | 350 | 300 | 5 | 200 |
| -26  | 855  | 435 | 100395 | 64897570 | 646.422 | 185 | 200 | 300 | 5 | 275 |
| -92  | 940  | 457 | 77030  | 48502600 | 629.659 | 189 | 250 | 200 | 5 | 200 |
| -82  | 643  | 320 | 73665  | 46140870 | 626.361 | 193 | 275 | 200 | 5 | 250 |
| -60  | 1059 | 621 | 54395  | 32610100 | 599.505 | 194 | 375 | 300 | 5 | 350 |
| -338 | 658  | 288 | 52963  | 31603331 | 596.706 | 201 | 375 | 300 | 5 | 225 |
| -188 | 1067 | 564 | 63410  | 38940590 | 614.108 | 202 | 325 | 300 | 5 | 275 |
| -281 | 576  | 278 | 90610  | 58037435 | 640.519 | 205 | 225 | 200 | 5 | 175 |
| -314 | 666  | 328 | 58715  | 35639205 | 606.986 | 213 | 375 | 300 | 5 | 250 |
| -27  | 670  | 346 | 70410  | 43855580 | 622.86  | 213 | 275 | 300 | 5 | 225 |
| 115  | 908  | 527 | 61305  | 37461510 | 611.068 | 215 | 350 | 300 | 5 | 275 |
| -109 | 759  | 290 | 76070  | 47823000 | 628.671 | 218 | 275 | 200 | 5 | 150 |
| -217 | 1001 | 454 | 54195  | 32472395 | 599.177 | 219 | 400 | 300 | 5 | 250 |
| 12   | 842  | 441 | 69835  | 43450600 | 622.189 | 222 | 300 | 300 | 5 | 325 |
| -113 | 740  | 359 | 68905  | 42794675 | 621.068 | 223 | 275 | 200 | 5 | 200 |
| -191 | 859  | 407 | 75040  | 47102980 | 627.705 | 223 | 275 | 200 | 5 | 275 |
| -94  | 855  | 422 | 59670  | 36308695 | 608.492 | 227 | 325 | 300 | 5 | 200 |
| -164 | 1110 | 496 | 61035  | 37272975 | 610.682 | 228 | 350 | 300 | 5 | 350 |
| -70  | 696  | 359 | 94640  | 60860705 | 643.076 | 231 | 225 | 200 | 5 | 125 |
| -229 | 527  | 240 | 73105  | 45745255 | 625.747 | 232 | 275 | 200 | 5 | 225 |
| -138 | 894  | 445 | 64210  | 39502625 | 615.21  | 233 | 300 | 200 | 5 | 300 |
| -180 | 1178 | 469 | 47495  | 28027960 | 590.124 | 233 | 400 | 300 | 5 | 325 |
| -191 | 614  | 283 | 83940  | 53348435 | 635.554 | 235 | 250 | 200 | 5 | 275 |
| -122 | 733  | 310 | 76880  | 48394980 | 629.487 | 238 | 250 | 200 | 5 | 125 |
| -117 | 630  | 280 | 65835  | 40641625 | 617.326 | 242 | 300 | 200 | 5 | 250 |
| -190 | 777  | 282 | 51125  | 30346010 | 593.565 | 242 | 400 | 300 | 5 | 275 |
| -283 | 649  | 335 | 48675  | 28737940 | 590.405 | 246 | 400 | 300 | 5 | 200 |
| -97  | 1008 | 366 | 81880  | 51906870 | 633.938 | 252 | 250 | 200 | 5 | 150 |
| -287 | 612  | 258 | 53345  | 31876170 | 597.547 | 262 | 400 | 300 | 5 | 225 |
| -250 | 900  | 310 | 66790  | 41315490 | 618.588 | 267 | 300 | 200 | 5 | 275 |
| -211 | 849  | 304 | 75900  | 47706880 | 628.549 | 273 | 250 | 200 | 5 | 100 |
| -92  | 676  | 328 | 74385  | 46647490 | 627.109 | 274 | 275 | 200 | 5 | 300 |
| -96  | 448  | 255 | 119635 | 78402090 | 655.344 | 286 | 175 | 200 | 5 | 150 |
| -271 | 631  | 246 | 62115  | 38027420 | 612.21  | 287 | 300 | 200 | 5 | 225 |
| -113 | 432  | 177 | 76798  | 48340671 | 629.452 | 290 | 250 | 200 | 5 | 175 |
| -289 | 593  | 265 | 66330  | 40988615 | 617.95  | 291 | 300 | 200 | 5 | 200 |

|      |      |     |        |          |         |     |     |     |   |     |
|------|------|-----|--------|----------|---------|-----|-----|-----|---|-----|
| -149 | 585  | 217 | 71675  | 44741435 | 624.227 | 294 | 275 | 200 | 5 | 125 |
| -299 | 470  | 192 | 58860  | 35748435 | 607.347 | 297 | 300 | 200 | 5 | 150 |
| -290 | 936  | 239 | 65710  | 40553895 | 617.165 | 303 | 300 | 200 | 5 | 100 |
| -157 | 878  | 377 | 49960  | 29590225 | 592.278 | 307 | 375 | 300 | 5 | 200 |
| -181 | 485  | 228 | 85665  | 54561150 | 636.913 | 313 | 225 | 200 | 5 | 150 |
| -240 | 441  | 233 | 70540  | 43943675 | 622.961 | 314 | 300 | 200 | 5 | 175 |
| -10  | 623  | 358 | 161853 | 1.08e+08 | 667.528 | 315 | 125 | 200 | 5 | 175 |
| -62  | 677  | 322 | 79330  | 50116915 | 631.752 | 349 | 275 | 200 | 5 | 175 |
| -345 | 1029 | 276 | 64570  | 39753670 | 615.668 | 357 | 300 | 200 | 5 | 125 |
| -369 | 2650 | 553 | 74471  | 46703977 | 627.143 | 451 | 275 | 200 | 5 | 100 |

## A.4 Random Ping Parameter Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| min | max   | avg   | msgs   | bytes   | bytes/msg | errors | SD  | K | TO1 | TO2 |
|-----|-------|-------|--------|---------|-----------|--------|-----|---|-----|-----|
| 71  | 41424 | 6434  | 510983 | 4625191 | 9.052     | 0      | 70  | 6 | 20  | 30  |
| 150 | 38390 | 7632  | 490499 | 4437652 | 9.047     | 0      | 70  | 4 | 30  | 20  |
| 79  | 53703 | 7755  | 510982 | 4619492 | 9.04      | 0      | 70  | 6 | 30  | 20  |
| 526 | 36286 | 7813  | 498745 | 4501629 | 9.026     | 0      | 70  | 4 | 20  | 30  |
| 189 | 30509 | 7845  | 514834 | 4651157 | 9.034     | 0      | 70  | 5 | 40  | 20  |
| 145 | 48914 | 7981  | 495743 | 4491596 | 9.06      | 0      | 70  | 5 | 30  | 30  |
| 111 | 65034 | 8120  | 471757 | 4285430 | 9.084     | 0      | 70  | 2 | 20  | 30  |
| 107 | 36120 | 8198  | 472924 | 4304731 | 9.102     | 0      | 70  | 3 | 30  | 20  |
| 101 | 55480 | 8384  | 506974 | 4587028 | 9.048     | 0      | 70  | 5 | 20  | 40  |
| 85  | 41330 | 8533  | 511692 | 4623559 | 9.036     | 0      | 70  | 1 | 40  | 20  |
| 212 | 55372 | 8727  | 519337 | 4690701 | 9.032     | 0      | 70  | 4 | 20  | 40  |
| 264 | 53263 | 9745  | 493788 | 4525944 | 9.166     | 0      | 100 | 6 | 30  | 50  |
| 131 | 59130 | 10587 | 479262 | 4411595 | 9.205     | 0      | 100 | 4 | 40  | 40  |
| 119 | 59773 | 10711 | 496425 | 4558277 | 9.182     | 0      | 100 | 6 | 20  | 60  |
| 325 | 61421 | 11125 | 485574 | 4455732 | 9.176     | 0      | 100 | 4 | 30  | 60  |
| 294 | 47048 | 11856 | 506253 | 4636587 | 9.159     | 0      | 100 | 6 | 30  | 60  |
| 305 | 75065 | 12336 | 517475 | 4725276 | 9.131     | 0      | 100 | 2 | 30  | 50  |
| 234 | 68132 | 15497 | 484738 | 4507813 | 9.299     | 0      | 150 | 1 | 30  | 100 |
| 482 | 84797 | 16126 | 479613 | 4462871 | 9.305     | 0      | 150 | 4 | 30  | 110 |

|     |        |       |        |         |       |   |     |   |    |     |
|-----|--------|-------|--------|---------|-------|---|-----|---|----|-----|
| 348 | 104769 | 16733 | 484243 | 4517011 | 9.328 | 0 | 150 | 4 | 30 | 100 |
| 306 | 82083  | 17079 | 484291 | 4504120 | 9.3   | 0 | 150 | 1 | 20 | 110 |
| 179 | 103980 | 17422 | 543692 | 5048804 | 9.286 | 0 | 150 | 5 | 30 | 100 |
| 168 | 83731  | 18270 | 466325 | 4346945 | 9.322 | 0 | 150 | 3 | 30 | 110 |
| 541 | 100521 | 18712 | 518887 | 4853269 | 9.353 | 0 | 200 | 3 | 40 | 140 |
| 209 | 107666 | 19184 | 540707 | 5013309 | 9.272 | 0 | 150 | 4 | 40 | 90  |
| 264 | 86208  | 20114 | 513692 | 4801134 | 9.346 | 0 | 200 | 3 | 40 | 150 |
| 211 | 84085  | 21269 | 504394 | 4717870 | 9.354 | 0 | 200 | 5 | 20 | 170 |
| 251 | 34642  | 7629  | 494637 | 4501668 | 9.101 | 1 | 70  | 5 | 30 | 20  |
| 529 | 46209  | 8693  | 503564 | 4563069 | 9.062 | 1 | 70  | 5 | 20 | 30  |
| 226 | 38377  | 8744  | 510455 | 4617035 | 9.045 | 1 | 70  | 6 | 30 | 30  |
| 75  | 49345  | 8752  | 507469 | 4588386 | 9.042 | 1 | 70  | 4 | 40 | 10  |
| 85  | 53706  | 9538  | 493777 | 4485446 | 9.084 | 1 | 70  | 6 | 40 | 10  |
| 137 | 39008  | 9548  | 515674 | 4673608 | 9.063 | 1 | 70  | 3 | 30 | 30  |
| 122 | 46143  | 9909  | 469938 | 4322035 | 9.197 | 1 | 100 | 5 | 20 | 60  |
| 147 | 45647  | 9968  | 542199 | 4905900 | 9.048 | 1 | 70  | 4 | 30 | 30  |
| 170 | 42174  | 10008 | 471707 | 4333826 | 9.188 | 1 | 100 | 2 | 40 | 50  |
| 248 | 61934  | 10157 | 486451 | 4442310 | 9.132 | 1 | 100 | 1 | 30 | 50  |
| 213 | 46912  | 10310 | 481300 | 4424267 | 9.192 | 1 | 100 | 5 | 20 | 70  |
| 123 | 54850  | 10348 | 474452 | 4351848 | 9.172 | 1 | 100 | 3 | 20 | 70  |
| 296 | 47814  | 10521 | 470479 | 4349987 | 9.246 | 1 | 100 | 6 | 40 | 50  |
| 268 | 69716  | 11258 | 472200 | 4336289 | 9.183 | 1 | 100 | 3 | 30 | 60  |
| 170 | 85522  | 13862 | 460098 | 4267539 | 9.275 | 1 | 150 | 2 | 20 | 110 |
| 387 | 73848  | 16658 | 498590 | 4624626 | 9.275 | 1 | 150 | 3 | 20 | 120 |
| 532 | 104052 | 17307 | 476241 | 4465004 | 9.376 | 1 | 200 | 4 | 30 | 150 |
| 299 | 84997  | 19932 | 486808 | 4559107 | 9.365 | 1 | 200 | 4 | 30 | 160 |
| 281 | 91457  | 21343 | 494986 | 4639589 | 9.373 | 1 | 200 | 6 | 20 | 160 |
| 457 | 86548  | 22665 | 514774 | 4812090 | 9.348 | 1 | 200 | 2 | 30 | 160 |
| 650 | 104639 | 23111 | 517585 | 4848385 | 9.367 | 1 | 200 | 4 | 40 | 140 |
| 217 | 119059 | 24729 | 540953 | 5050667 | 9.337 | 1 | 200 | 6 | 30 | 150 |
| 123 | 34787  | 6215  | 479070 | 4352571 | 9.085 | 2 | 70  | 2 | 40 | 10  |
| 160 | 51239  | 7240  | 515725 | 4654970 | 9.026 | 2 | 70  | 3 | 40 | 10  |
| 217 | 39735  | 7712  | 483030 | 4374959 | 9.057 | 2 | 70  | 1 | 40 | 10  |
| 90  | 39886  | 8110  | 498843 | 4529426 | 9.08  | 2 | 70  | 4 | 40 | 20  |
| 308 | 53905  | 8393  | 514677 | 4641428 | 9.018 | 2 | 70  | 3 | 20 | 40  |
| 409 | 36737  | 8436  | 459223 | 4188143 | 9.12  | 2 | 70  | 1 | 30 | 30  |
| 179 | 46036  | 9425  | 499328 | 4578214 | 9.169 | 2 | 100 | 4 | 30 | 50  |

|     |        |       |        |         |       |   |     |   |    |     |
|-----|--------|-------|--------|---------|-------|---|-----|---|----|-----|
| 286 | 55237  | 10962 | 494516 | 4529831 | 9.16  | 2 | 100 | 3 | 30 | 50  |
| 130 | 41305  | 11178 | 500533 | 4591923 | 9.174 | 2 | 100 | 5 | 40 | 40  |
| 184 | 56368  | 14569 | 461349 | 4297167 | 9.314 | 2 | 150 | 5 | 20 | 120 |
| 448 | 53912  | 14839 | 468781 | 4363363 | 9.308 | 2 | 150 | 3 | 40 | 90  |
| 269 | 67717  | 14889 | 467802 | 4369047 | 9.34  | 2 | 150 | 6 | 40 | 100 |
| 236 | 66597  | 15099 | 468267 | 4359737 | 9.31  | 2 | 150 | 4 | 20 | 120 |
| 315 | 96267  | 15549 | 515517 | 4794284 | 9.3   | 2 | 150 | 4 | 40 | 100 |
| 189 | 72841  | 16000 | 484372 | 4516080 | 9.324 | 2 | 150 | 6 | 30 | 100 |
| 303 | 78050  | 17977 | 494978 | 4597756 | 9.289 | 2 | 150 | 6 | 20 | 120 |
| 214 | 101100 | 21202 | 529009 | 4938889 | 9.336 | 2 | 200 | 1 | 40 | 140 |
| 540 | 88022  | 22331 | 516382 | 4831509 | 9.356 | 2 | 200 | 2 | 40 | 140 |
| 230 | 110102 | 22463 | 519527 | 4854830 | 9.345 | 2 | 200 | 4 | 40 | 150 |
| 176 | 53139  | 7260  | 470486 | 4283482 | 9.104 | 3 | 70  | 3 | 20 | 30  |
| 141 | 40056  | 7932  | 470727 | 4281869 | 9.096 | 3 | 70  | 2 | 30 | 20  |
| 74  | 37583  | 8620  | 526606 | 4753864 | 9.027 | 3 | 70  | 6 | 40 | 20  |
| 143 | 32340  | 8795  | 468452 | 4311418 | 9.204 | 3 | 100 | 6 | 20 | 70  |
| 131 | 54268  | 9289  | 481774 | 4418947 | 9.172 | 3 | 100 | 4 | 20 | 60  |
| 347 | 52499  | 10231 | 462880 | 4269306 | 9.223 | 3 | 100 | 1 | 40 | 40  |
| 123 | 61843  | 11010 | 478882 | 4400485 | 9.189 | 3 | 100 | 1 | 20 | 60  |
| 322 | 77222  | 14269 | 478199 | 4444186 | 9.294 | 3 | 150 | 2 | 20 | 120 |
| 197 | 60884  | 15702 | 485547 | 4500891 | 9.27  | 3 | 150 | 1 | 40 | 100 |
| 164 | 67108  | 16176 | 482788 | 4503420 | 9.328 | 3 | 150 | 3 | 30 | 100 |
| 337 | 107535 | 21268 | 503470 | 4722268 | 9.379 | 3 | 200 | 5 | 20 | 160 |
| 247 | 112876 | 21688 | 510260 | 4767754 | 9.344 | 3 | 200 | 1 | 30 | 160 |
| 239 | 28795  | 7448  | 498423 | 4531260 | 9.091 | 4 | 70  | 5 | 40 | 10  |
| 185 | 46119  | 8226  | 492245 | 4457912 | 9.056 | 4 | 70  | 2 | 40 | 20  |
| 393 | 73587  | 12964 | 478603 | 4400847 | 9.195 | 4 | 100 | 3 | 40 | 40  |
| 233 | 85673  | 15635 | 470124 | 4375423 | 9.307 | 4 | 150 | 5 | 30 | 110 |
| 193 | 79915  | 16122 | 464067 | 4316304 | 9.301 | 4 | 150 | 2 | 40 | 90  |
| 218 | 109766 | 16479 | 474943 | 4417013 | 9.3   | 4 | 150 | 6 | 20 | 110 |
| 477 | 75367  | 17588 | 508731 | 4722392 | 9.283 | 4 | 150 | 3 | 20 | 110 |
| 403 | 78268  | 19136 | 519887 | 4869524 | 9.367 | 4 | 200 | 5 | 40 | 140 |
| 310 | 110397 | 19854 | 488681 | 4585061 | 9.383 | 4 | 200 | 1 | 30 | 150 |
| 764 | 119982 | 25239 | 534019 | 4996170 | 9.356 | 4 | 200 | 6 | 30 | 160 |
| 181 | 30637  | 7481  | 466710 | 4296543 | 9.206 | 5 | 100 | 3 | 20 | 60  |
| 260 | 36027  | 8312  | 495699 | 4493730 | 9.065 | 5 | 70  | 6 | 20 | 40  |
| 129 | 46461  | 8545  | 498437 | 4509508 | 9.047 | 5 | 70  | 1 | 30 | 20  |



|     |        |       |        |         |       |    |     |   |    |     |
|-----|--------|-------|--------|---------|-------|----|-----|---|----|-----|
| 124 | 38414  | 9266  | 463454 | 4259000 | 9.19  | 5  | 100 | 1 | 20 | 70  |
| 230 | 60622  | 10526 | 462700 | 4275015 | 9.239 | 5  | 100 | 2 | 30 | 60  |
| 209 | 56533  | 10780 | 477466 | 4389550 | 9.193 | 5  | 100 | 2 | 20 | 70  |
| 115 | 61583  | 11483 | 461058 | 4246969 | 9.211 | 5  | 100 | 2 | 20 | 60  |
| 265 | 108497 | 17907 | 490777 | 4545225 | 9.261 | 5  | 150 | 1 | 30 | 110 |
| 203 | 63472  | 10477 | 480878 | 4416484 | 9.184 | 6  | 100 | 1 | 40 | 50  |
| 163 | 45447  | 10658 | 485436 | 4463245 | 9.194 | 6  | 100 | 5 | 30 | 50  |
| 413 | 90964  | 16518 | 452210 | 4215402 | 9.322 | 6  | 150 | 6 | 30 | 110 |
| 465 | 93619  | 17286 | 520190 | 4866160 | 9.355 | 6  | 150 | 5 | 40 | 100 |
| 549 | 104641 | 19456 | 518711 | 4894918 | 9.437 | 6  | 200 | 4 | 20 | 170 |
| 248 | 87773  | 19900 | 512609 | 4809840 | 9.383 | 6  | 200 | 6 | 40 | 140 |
| 264 | 93010  | 20184 | 491107 | 4594552 | 9.356 | 6  | 200 | 3 | 20 | 160 |
| 298 | 114765 | 21738 | 512478 | 4808914 | 9.384 | 6  | 200 | 6 | 40 | 150 |
| 432 | 58734  | 16008 | 490894 | 4564220 | 9.298 | 7  | 150 | 4 | 20 | 110 |
| 616 | 117432 | 17828 | 492755 | 4616397 | 9.369 | 7  | 200 | 2 | 40 | 150 |
| 88  | 33630  | 7914  | 510482 | 4613459 | 9.037 | 8  | 70  | 3 | 40 | 20  |
| 322 | 40881  | 11011 | 466114 | 4293503 | 9.211 | 8  | 100 | 1 | 30 | 60  |
| 670 | 94091  | 18450 | 484164 | 4533407 | 9.363 | 8  | 200 | 3 | 20 | 170 |
| 308 | 122429 | 19955 | 490146 | 4576280 | 9.337 | 8  | 200 | 3 | 30 | 150 |
| 391 | 35174  | 11548 | 483064 | 4441141 | 9.194 | 9  | 100 | 4 | 20 | 70  |
| 656 | 82676  | 20962 | 503793 | 4704230 | 9.338 | 9  | 200 | 3 | 30 | 160 |
| 265 | 80229  | 22008 | 549934 | 5130753 | 9.33  | 9  | 200 | 1 | 40 | 150 |
| 256 | 84742  | 22144 | 505340 | 4720680 | 9.342 | 9  | 200 | 1 | 20 | 160 |
| 155 | 94736  | 14653 | 475219 | 4430404 | 9.323 | 10 | 150 | 2 | 30 | 100 |
| 114 | 36909  | 8252  | 489062 | 4436133 | 9.071 | 11 | 70  | 1 | 20 | 40  |
| 196 | 31006  | 8547  | 507654 | 4591332 | 9.044 | 12 | 70  | 2 | 30 | 30  |
| 184 | 49054  | 9649  | 505583 | 4634510 | 9.167 | 12 | 100 | 6 | 40 | 40  |
| 429 | 69025  | 13890 | 487901 | 4541774 | 9.309 | 12 | 150 | 5 | 40 | 90  |
| 355 | 90907  | 25548 | 531278 | 4961137 | 9.338 | 12 | 200 | 5 | 30 | 150 |
| 278 | 49578  | 8578  | 480501 | 4360388 | 9.075 | 14 | 70  | 1 | 20 | 30  |
| 403 | 43649  | 10276 | 472921 | 4345042 | 9.188 | 14 | 100 | 5 | 40 | 50  |
| 669 | 51633  | 15548 | 470416 | 4381601 | 9.314 | 14 | 150 | 2 | 30 | 110 |
| 262 | 82935  | 19237 | 500219 | 4680688 | 9.357 | 14 | 200 | 6 | 20 | 170 |
| 224 | 120588 | 19230 | 489363 | 4585639 | 9.371 | 15 | 200 | 2 | 30 | 150 |
| 198 | 42756  | 11159 | 462352 | 4253881 | 9.201 | 16 | 100 | 2 | 40 | 40  |
| 220 | 111661 | 23946 | 510298 | 4789546 | 9.386 | 16 | 200 | 4 | 20 | 160 |
| 233 | 97636  | 22114 | 539059 | 5035136 | 9.341 | 18 | 200 | 5 | 40 | 150 |

|     |        |       |        |         |       |    |     |   |    |     |
|-----|--------|-------|--------|---------|-------|----|-----|---|----|-----|
| 223 | 83890  | 14305 | 479734 | 4461558 | 9.3   | 20 | 150 | 6 | 40 | 90  |
| 434 | 56477  | 11424 | 492455 | 4513068 | 9.164 | 21 | 100 | 3 | 40 | 50  |
| 235 | 82771  | 21529 | 504505 | 4730828 | 9.377 | 21 | 200 | 5 | 30 | 160 |
| 211 | 93952  | 16745 | 500131 | 4632251 | 9.262 | 23 | 150 | 1 | 20 | 120 |
| 325 | 78348  | 14969 | 472975 | 4385939 | 9.273 | 26 | 150 | 1 | 40 | 90  |
| 174 | 100720 | 15070 | 477947 | 4439949 | 9.29  | 26 | 150 | 2 | 40 | 100 |
| 270 | 102561 | 23705 | 497031 | 4654009 | 9.364 | 29 | 200 | 1 | 20 | 170 |
| 219 | 45363  | 13051 | 455084 | 4264436 | 9.371 | 30 | 150 | 5 | 20 | 110 |
| 153 | 95270  | 11329 | 498538 | 4559831 | 9.146 | 32 | 100 | 4 | 40 | 50  |
| 584 | 73706  | 17615 | 493734 | 4619973 | 9.357 | 32 | 200 | 2 | 20 | 160 |
| 444 | 85386  | 20696 | 501238 | 4672906 | 9.323 | 38 | 200 | 2 | 20 | 170 |
| 146 | 49977  | 9861  | 502995 | 4598454 | 9.142 | 39 | 100 | 5 | 30 | 60  |
| 138 | 49820  | 9404  | 511170 | 4617233 | 9.033 | 63 | 70  | 2 | 20 | 40  |

## A.5 Aggressive Parameter Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

The chosen set of parameters were SD = 100, K = 4, TO1 = 20, and TO2 = 60. They were picked because that run did not have a negative minimum detection time and it had a value of  $k$  that was equal to the value of  $k$  found for the Random Ping protocol. Having  $k$  equal would be helpful when examining the effects of the modifications made in the Aggressive protocol. This would eliminate the possibility of performance differences caused by different values of  $k$ .

| min    | max  | avg   | msgs   | bytes   | bytes/msgerrors | SD | K   | TO1 | TO2 |     |
|--------|------|-------|--------|---------|-----------------|----|-----|-----|-----|-----|
| -2162  | 506  | -1611 | 470719 | 4193093 | 8.908           | 0  | 150 | 3   | 40  | 100 |
| -2091  | 404  | -1038 | 744443 | 6639759 | 8.919           | 0  | 70  | 4   | 40  | 10  |
| -1150  | 1015 | -603  | 589158 | 5287154 | 8.974           | 0  | 100 | 6   | 40  | 50  |
| -14222 | 232  | 84    | 522585 | 4638402 | 8.876           | 0  | 150 | 4   | 40  | 100 |
| 94     | 160  | 101   | 524193 | 4581189 | 8.74            | 0  | 70  | 1   | 20  | 30  |
| 103    | 162  | 108   | 581941 | 5186747 | 8.913           | 0  | 100 | 3   | 40  | 50  |
| -573   | 160  | 116   | 515547 | 4549190 | 8.824           | 0  | 100 | 1   | 20  | 60  |
| -15258 | 1541 | 129   | 508613 | 4554253 | 8.954           | 0  | 200 | 3   | 30  | 160 |
| 125    | 304  | 135   | 551660 | 4897027 | 8.877           | 0  | 100 | 4   | 20  | 60  |
| 146    | 228  | 157   | 579467 | 5136305 | 8.864           | 0  | 100 | 4   | 30  | 50  |

|       |      |       |        |         |       |   |     |   |    |     |
|-------|------|-------|--------|---------|-------|---|-----|---|----|-----|
| 170   | 177  | 173   | 497545 | 4421692 | 8.887 | 0 | 150 | 4 | 40 | 90  |
| 110   | 858  | 177   | 542045 | 4785192 | 8.828 | 0 | 70  | 1 | 20 | 40  |
| 173   | 203  | 180   | 575998 | 5099635 | 8.854 | 0 | 100 | 4 | 30 | 60  |
| 178   | 299  | 192   | 549810 | 4874005 | 8.865 | 0 | 100 | 1 | 20 | 70  |
| 186   | 197  | 192   | 647771 | 5763994 | 8.898 | 0 | 70  | 4 | 20 | 30  |
| 189   | 218  | 193   | 569457 | 5051471 | 8.871 | 0 | 100 | 2 | 30 | 50  |
| 213   | 222  | 216   | 485554 | 4356010 | 8.971 | 0 | 200 | 1 | 30 | 160 |
| 214   | 222  | 218   | 498845 | 4453726 | 8.928 | 0 | 200 | 4 | 30 | 150 |
| 234   | 242  | 238   | 510474 | 4579038 | 8.97  | 0 | 200 | 5 | 20 | 170 |
| -5494 | 302  | 240   | 504524 | 4487903 | 8.895 | 0 | 150 | 2 | 20 | 110 |
| -306  | 7374 | 243   | 558721 | 4910892 | 8.79  | 0 | 70  | 3 | 40 | 20  |
| 241   | 272  | 247   | 509473 | 4553323 | 8.937 | 0 | 150 | 2 | 20 | 120 |
| 254   | 262  | 258   | 504551 | 4479637 | 8.878 | 0 | 150 | 1 | 20 | 110 |
| 258   | 265  | 261   | 493107 | 4420196 | 8.964 | 0 | 200 | 1 | 20 | 170 |
| 274   | 283  | 279   | 572988 | 5086452 | 8.877 | 0 | 100 | 4 | 40 | 50  |
| 285   | 291  | 288   | 510658 | 4583607 | 8.976 | 0 | 200 | 4 | 20 | 170 |
| 278   | 464  | 290   | 494839 | 4441794 | 8.976 | 0 | 200 | 3 | 20 | 170 |
| 287   | 296  | 291   | 470745 | 4200382 | 8.923 | 0 | 200 | 1 | 30 | 150 |
| 294   | 301  | 297   | 514941 | 4584932 | 8.904 | 0 | 150 | 4 | 20 | 110 |
| 285   | 520  | 301   | 483640 | 4310700 | 8.913 | 0 | 200 | 3 | 30 | 150 |
| 304   | 482  | 312   | 528438 | 4722514 | 8.937 | 0 | 150 | 4 | 20 | 120 |
| 303   | 417  | 316   | 512452 | 4585789 | 8.949 | 0 | 150 | 3 | 20 | 120 |
| 365   | 374  | 369   | 533367 | 4757482 | 8.92  | 0 | 150 | 5 | 20 | 120 |
| 446   | 552  | 471   | 492605 | 4420993 | 8.975 | 0 | 200 | 3 | 40 | 150 |
| 649   | 675  | 661   | 488379 | 4350076 | 8.907 | 0 | 150 | 1 | 40 | 100 |
| 748   | 757  | 752   | 560395 | 4915118 | 8.771 | 0 | 70  | 2 | 20 | 30  |
| -7564 | 4527 | 1499  | 501575 | 4503361 | 8.978 | 0 | 200 | 6 | 40 | 150 |
| -4396 | 231  | -1715 | 559091 | 4960643 | 8.873 | 1 | 100 | 5 | 30 | 50  |
| -9493 | 5355 | 750   | 521275 | 4637710 | 8.897 | 1 | 150 | 4 | 30 | 110 |
| 206   | 213  | 209   | 494567 | 4408445 | 8.914 | 2 | 200 | 3 | 40 | 140 |
| 332   | 348  | 337   | 516192 | 4591400 | 8.895 | 3 | 150 | 3 | 20 | 110 |
| 408   | 415  | 411   | 496728 | 4401697 | 8.861 | 3 | 150 | 6 | 40 | 90  |
| 222   | 229  | 226   | 485832 | 4338051 | 8.929 | 4 | 200 | 1 | 20 | 160 |
| -515  | 660  | 630   | 497201 | 4438432 | 8.927 | 5 | 200 | 3 | 20 | 160 |
| -8869 | 492  | -530  | 539978 | 4783098 | 8.858 | 6 | 100 | 2 | 20 | 70  |
| 99    | 479  | 131   | 613981 | 5421974 | 8.831 | 6 | 70  | 6 | 30 | 30  |
| 294   | 462  | 318   | 483438 | 4325275 | 8.947 | 8 | 200 | 2 | 30 | 150 |

|        |       |       |        |         |       |    |     |   |    |     |
|--------|-------|-------|--------|---------|-------|----|-----|---|----|-----|
| 728    | 895   | 747   | 504097 | 4523979 | 8.974 | 8  | 200 | 5 | 40 | 150 |
| -4518  | 1896  | -94   | 579153 | 5097560 | 8.802 | 9  | 70  | 3 | 20 | 30  |
| 445    | 555   | 460   | 629337 | 5490279 | 8.724 | 9  | 70  | 3 | 30 | 30  |
| -4317  | 697   | -1674 | 536962 | 4766899 | 8.878 | 10 | 100 | 4 | 40 | 40  |
| 145    | 206   | 152   | 571997 | 5079393 | 8.88  | 10 | 100 | 5 | 40 | 50  |
| -2585  | 502   | -1506 | 576947 | 5119195 | 8.873 | 14 | 70  | 6 | 20 | 40  |
| 449    | 457   | 454   | 550614 | 4875187 | 8.854 | 19 | 100 | 1 | 40 | 50  |
| 336    | 436   | 361   | 513159 | 4530607 | 8.829 | 21 | 100 | 6 | 20 | 70  |
| 390    | 399   | 394   | 471469 | 4226717 | 8.965 | 23 | 200 | 5 | 30 | 150 |
| -2246  | 261   | -1203 | 572240 | 5017728 | 8.769 | 24 | 70  | 2 | 40 | 20  |
| -3717  | 288   | -1093 | 563444 | 4977229 | 8.834 | 25 | 100 | 1 | 40 | 40  |
| 88     | 16566 | 346   | 507767 | 4528034 | 8.918 | 27 | 150 | 6 | 30 | 110 |
| 153    | 217   | 161   | 581503 | 5160356 | 8.874 | 29 | 100 | 1 | 30 | 60  |
| 196    | 204   | 201   | 521945 | 4591227 | 8.796 | 29 | 100 | 3 | 20 | 70  |
| -3943  | 7475  | 1296  | 506703 | 4496351 | 8.874 | 30 | 150 | 3 | 30 | 100 |
| 147    | 222   | 159   | 536420 | 4748625 | 8.852 | 34 | 100 | 5 | 20 | 70  |
| -8982  | 6903  | 446   | 498420 | 4412505 | 8.853 | 34 | 150 | 1 | 30 | 100 |
| -18993 | 591   | 376   | 489119 | 4390744 | 8.977 | 35 | 200 | 4 | 40 | 150 |
| -530   | 601   | 447   | 656674 | 5873141 | 8.944 | 35 | 100 | 5 | 20 | 60  |
| 79     | 179   | 89    | 539485 | 4742086 | 8.79  | 37 | 70  | 5 | 40 | 10  |
| -3673  | 838   | -1482 | 462529 | 4131954 | 8.933 | 38 | 150 | 1 | 40 | 90  |
| -4073  | 4155  | -357  | 494024 | 4419690 | 8.946 | 39 | 200 | 5 | 40 | 140 |
| 130    | 619   | 149   | 559231 | 4898073 | 8.759 | 41 | 70  | 1 | 30 | 30  |
| 259    | 267   | 263   | 490041 | 4360742 | 8.899 | 41 | 150 | 6 | 30 | 100 |
| 335    | 343   | 339   | 625775 | 5562100 | 8.888 | 41 | 100 | 6 | 30 | 60  |
| 610    | 620   | 614   | 651817 | 5778264 | 8.865 | 41 | 70  | 1 | 30 | 20  |
| -5722  | 1104  | -37   | 542763 | 4828276 | 8.896 | 42 | 70  | 2 | 20 | 40  |
| 298    | 478   | 314   | 510430 | 4547850 | 8.91  | 42 | 150 | 5 | 30 | 110 |
| -205   | 311   | -83   | 727297 | 6417274 | 8.823 | 43 | 70  | 3 | 30 | 20  |
| 158    | 265   | 165   | 585073 | 5165167 | 8.828 | 43 | 70  | 3 | 20 | 40  |
| -277   | 2148  | 238   | 636923 | 5715502 | 8.974 | 43 | 100 | 5 | 40 | 40  |
| -665   | 9122  | 1003  | 517968 | 4592840 | 8.867 | 45 | 100 | 3 | 20 | 60  |
| -5597  | 2850  | 1112  | 908424 | 8226358 | 9.056 | 45 | 70  | 6 | 40 | 10  |
| 120    | 132   | 125   | 585610 | 5186651 | 8.857 | 47 | 70  | 2 | 30 | 20  |
| -10149 | 5367  | 1485  | 494788 | 4440406 | 8.974 | 48 | 200 | 5 | 30 | 160 |
| 96     | 157   | 107   | 547047 | 4815368 | 8.802 | 49 | 70  | 1 | 40 | 20  |
| 103    | 305   | 109   | 592158 | 5261101 | 8.885 | 50 | 100 | 3 | 30 | 50  |

|        |      |       |        |         |       |     |     |   |    |     |
|--------|------|-------|--------|---------|-------|-----|-----|---|----|-----|
| -5526  | 239  | 150   | 563448 | 4983629 | 8.845 | 50  | 100 | 3 | 30 | 60  |
| 127    | 264  | 145   | 537182 | 4753590 | 8.849 | 54  | 100 | 4 | 20 | 70  |
| 194    | 201  | 197   | 527937 | 4711182 | 8.924 | 54  | 150 | 6 | 20 | 120 |
| -9477  | 3432 | 1230  | 504315 | 4516786 | 8.956 | 57  | 200 | 2 | 20 | 170 |
| -898   | 762  | -432  | 506114 | 4518699 | 8.928 | 61  | 150 | 5 | 20 | 110 |
| 818    | 826  | 822   | 503293 | 4523474 | 8.988 | 63  | 200 | 6 | 20 | 170 |
| 488    | 500  | 492   | 495479 | 4435015 | 8.951 | 64  | 200 | 4 | 30 | 160 |
| -5407  | 3317 | 114   | 493935 | 4416876 | 8.942 | 65  | 200 | 4 | 40 | 140 |
| 359    | 378  | 363   | 495150 | 4411066 | 8.909 | 65  | 150 | 5 | 30 | 100 |
| -1948  | 6592 | 700   | 516635 | 4624716 | 8.952 | 68  | 200 | 2 | 30 | 160 |
| -5611  | 128  | -974  | 631724 | 5599711 | 8.864 | 69  | 70  | 4 | 30 | 30  |
| -538   | 703  | -343  | 569977 | 5095887 | 8.941 | 69  | 100 | 2 | 30 | 60  |
| 127    | 730  | 189   | 574890 | 5119183 | 8.905 | 69  | 100 | 5 | 30 | 60  |
| -4443  | 1129 | -322  | 657650 | 5799986 | 8.819 | 71  | 70  | 5 | 30 | 30  |
| -7300  | 581  | 180   | 584543 | 5118820 | 8.757 | 73  | 70  | 1 | 40 | 10  |
| 448    | 459  | 452   | 516052 | 4591817 | 8.898 | 73  | 150 | 1 | 30 | 110 |
| 178    | 259  | 185   | 567859 | 5013608 | 8.829 | 75  | 100 | 6 | 20 | 60  |
| -5586  | 238  | 90    | 521955 | 4623567 | 8.858 | 76  | 100 | 2 | 20 | 60  |
| 767    | 775  | 771   | 555487 | 4894056 | 8.81  | 77  | 100 | 3 | 40 | 40  |
| -6540  | 8966 | 1214  | 508652 | 4517832 | 8.882 | 80  | 150 | 2 | 30 | 110 |
| 138    | 406  | 164   | 478347 | 4255559 | 8.896 | 81  | 150 | 2 | 40 | 90  |
| -5055  | 262  | -807  | 586088 | 5192700 | 8.86  | 83  | 70  | 5 | 40 | 20  |
| 200    | 238  | 204   | 518558 | 4617638 | 8.905 | 87  | 150 | 6 | 40 | 100 |
| 139    | 341  | 148   | 646613 | 5726777 | 8.857 | 88  | 70  | 5 | 20 | 40  |
| -1697  | 5211 | 513   | 493638 | 4442349 | 8.999 | 88  | 200 | 6 | 30 | 160 |
| 390    | 401  | 396   | 604146 | 5328300 | 8.82  | 89  | 70  | 2 | 30 | 30  |
| -7078  | 614  | -2906 | 508092 | 4531453 | 8.919 | 96  | 150 | 5 | 40 | 100 |
| -1288  | 255  | 235   | 495559 | 4408009 | 8.895 | 96  | 150 | 2 | 30 | 100 |
| 110    | 365  | 126   | 550065 | 4890924 | 8.892 | 102 | 100 | 1 | 30 | 50  |
| -557   | 308  | -383  | 544168 | 4833457 | 8.882 | 103 | 100 | 6 | 40 | 40  |
| 511    | 520  | 514   | 494112 | 4428708 | 8.963 | 111 | 200 | 2 | 40 | 150 |
| 389    | 396  | 392   | 587288 | 5176278 | 8.814 | 115 | 70  | 4 | 20 | 40  |
| 188    | 546  | 207   | 504979 | 4486300 | 8.884 | 119 | 150 | 5 | 40 | 90  |
| 100    | 291  | 112   | 571778 | 5044526 | 8.823 | 127 | 70  | 3 | 40 | 10  |
| 145    | 179  | 152   | 692272 | 6247176 | 9.024 | 133 | 100 | 6 | 30 | 50  |
| 112    | 214  | 130   | 575520 | 5050533 | 8.776 | 137 | 70  | 6 | 40 | 20  |
| -10361 | 563  | 224   | 604495 | 5374428 | 8.891 | 141 | 100 | 2 | 40 | 50  |

|        |      |       |        |         |       |      |     |   |    |     |
|--------|------|-------|--------|---------|-------|------|-----|---|----|-----|
| 257    | 267  | 261   | 476638 | 4253777 | 8.925 | 156  | 200 | 1 | 40 | 140 |
| 370    | 443  | 379   | 498587 | 4458632 | 8.943 | 246  | 200 | 6 | 20 | 160 |
| -5442  | 331  | 269   | 487597 | 4357147 | 8.936 | 280  | 200 | 6 | 30 | 150 |
| -4050  | 3662 | 258   | 613453 | 5428117 | 8.848 | 304  | 70  | 4 | 30 | 20  |
| 250    | 266  | 254   | 509471 | 4538139 | 8.908 | 326  | 200 | 5 | 20 | 160 |
| 476    | 494  | 482   | 463575 | 4143506 | 8.938 | 330  | 200 | 2 | 40 | 140 |
| 573    | 582  | 577   | 473679 | 4226159 | 8.922 | 340  | 200 | 2 | 20 | 160 |
| 80     | 959  | 128   | 716716 | 6405279 | 8.937 | 341  | 70  | 6 | 20 | 30  |
| -5896  | 1808 | 29    | 557834 | 4886638 | 8.76  | 375  | 70  | 4 | 40 | 20  |
| -397   | 1260 | 73    | 707897 | 6363999 | 8.99  | 395  | 70  | 5 | 20 | 30  |
| -5315  | 241  | 73    | 539108 | 4774919 | 8.857 | 405  | 100 | 2 | 40 | 40  |
| 292    | 304  | 296   | 494359 | 4406221 | 8.913 | 487  | 200 | 4 | 20 | 160 |
| 332    | 340  | 335   | 492192 | 4410407 | 8.961 | 499  | 200 | 1 | 40 | 150 |
| -10047 | 1259 | -1683 | 550331 | 4906903 | 8.916 | 527  | 150 | 6 | 20 | 110 |
| 233    | 246  | 239   | 471524 | 4198531 | 8.904 | 530  | 150 | 2 | 40 | 100 |
| -9927  | 752  | -2452 | 510226 | 4540085 | 8.898 | 576  | 150 | 3 | 40 | 90  |
| 213    | 239  | 217   | 500156 | 4463074 | 8.923 | 621  | 150 | 3 | 30 | 110 |
| 152    | 171  | 159   | 511998 | 4536538 | 8.86  | 703  | 150 | 4 | 30 | 100 |
| 3443   | 3452 | 3447  | 546664 | 4868138 | 8.905 | 922  | 200 | 6 | 40 | 140 |
| 122    | 511  | 141   | 599805 | 5307609 | 8.849 | 1236 | 70  | 2 | 40 | 10  |

## B Group Size Data

nodes = The total number of nodes in the group.

min = The minimum time for a node to detect the failure.

max = The maximum time for a node to detect the failure.

avg = The average time for a node to detect the failure.

msgs = The total number of messages sent by the nodes.

bytes = The total number of bytes sent by the nodes.

bytes/msg = The average number of bytes per message sent.

errors = The total number of false detections.

### B.1 Heartbeat Group Size Data

SD = The time interval between heartbeat messages.

TO = The Heartbeat timeout value.

CD = The time interval between scans for failed nodes.

| nodes | min | max | avg | msgs | bytes | bytes/msg | errors | SD | TO | CD |
|-------|-----|-----|-----|------|-------|-----------|--------|----|----|----|
|-------|-----|-----|-----|------|-------|-----------|--------|----|----|----|

|     |       |       |       |        |         |       |     |     |     |     |
|-----|-------|-------|-------|--------|---------|-------|-----|-----|-----|-----|
| 10  | 439   | 679   | 508   | 10600  | 44520   | 4.2   | 0   | 650 | 700 | 700 |
| 20  | 581   | 770   | 687   | 41600  | 189280  | 4.55  | 0   | 650 | 700 | 700 |
| 30  | 319   | 979   | 554   | 90900  | 427230  | 4.7   | 0   | 650 | 700 | 700 |
| 40  | 261   | 919   | 591   | 158080 | 755080  | 4.777 | 0   | 650 | 700 | 700 |
| 50  | 352   | 1050  | 628   | 246900 | 1190400 | 4.821 | 0   | 650 | 700 | 700 |
| 60  | 652   | 1360  | 983   | 345660 | 1676460 | 4.85  | 19  | 650 | 700 | 700 |
| 70  | 355   | 1061  | 803   | 439810 | 2142840 | 4.872 | 0   | 650 | 700 | 700 |
| 80  | 510   | 1327  | 850   | 521920 | 2551760 | 4.889 | 0   | 650 | 700 | 700 |
| 90  | 341   | 1010  | 776   | 596246 | 2922550 | 4.902 | 0   | 650 | 700 | 700 |
| 100 | 439   | 1212  | 901   | 623968 | 3076640 | 4.931 | 2   | 650 | 700 | 700 |
| 110 | 122   | 819   | 353   | 610720 | 3069330 | 5.026 | 0   | 650 | 700 | 700 |
| 120 | 590   | 1270  | 800   | 614446 | 3135664 | 5.103 | 4   | 650 | 700 | 700 |
| 130 | 412   | 1402  | 668   | 690300 | 3570580 | 5.173 | 0   | 650 | 700 | 700 |
| 140 | 322   | 1135  | 589   | 709302 | 3722070 | 5.248 | 13  | 650 | 700 | 700 |
| 150 | 339   | 1303  | 606   | 513637 | 2715372 | 5.287 | 0   | 650 | 700 | 700 |
| 160 | 509   | 1150  | 718   | 301110 | 1606096 | 5.334 | 0   | 650 | 700 | 700 |
| 170 | 122   | 524   | 346   | 239697 | 1312212 | 5.474 | 26  | 650 | 700 | 700 |
| 180 | 3866  | 5402  | 4129  | 138482 | 762165  | 5.504 | 6   | 650 | 700 | 700 |
| 190 | 9863  | 11637 | 10155 | 121244 | 643797  | 5.31  | 27  | 650 | 700 | 700 |
| 200 | 22519 | 25149 | 22885 | 142545 | 769065  | 5.395 | 366 | 650 | 700 | 700 |

## B.2 Ping Group Size Data

SD = The time interval between ping messages.

ACK\_TO = The timeout value for a ping acknowledgment message.

| nodes | min  | max  | avg  | msgs   | bytes   | bytes/msg | errors | SD   | ACK_TO |
|-------|------|------|------|--------|---------|-----------|--------|------|--------|
| 10    | 1209 | 1339 | 1275 | 13970  | 86614   | 6.2       | 0      | 1000 | 400    |
| 20    | 474  | 1329 | 893  | 54770  | 358818  | 6.551     | 0      | 1000 | 400    |
| 30    | 415  | 1373 | 778  | 118700 | 795307  | 6.7       | 0      | 1000 | 400    |
| 40    | 499  | 1115 | 844  | 201480 | 1365027 | 6.775     | 0      | 1000 | 400    |
| 50    | 630  | 1041 | 786  | 318860 | 2174537 | 6.82      | 0      | 1000 | 400    |
| 60    | 495  | 1365 | 845  | 453182 | 3104397 | 6.85      | 0      | 1000 | 400    |
| 70    | 430  | 1410 | 674  | 597024 | 4102775 | 6.872     | 0      | 1000 | 400    |
| 80    | 441  | 1424 | 1009 | 711801 | 4902913 | 6.888     | 0      | 1000 | 400    |
| 90    | 436  | 1060 | 777  | 792990 | 5471514 | 6.9       | 0      | 1000 | 400    |
| 100   | 692  | 1356 | 1091 | 935694 | 6485110 | 6.931     | 0      | 1000 | 400    |
| 110   | 38   | 1518 | 1053 | 987952 | 6934693 | 7.019     | 18     | 1000 | 400    |

|     |       |       |       |         |         |       |     |      |     |
|-----|-------|-------|-------|---------|---------|-------|-----|------|-----|
| 120 | 417   | 1532  | 1154  | 1003338 | 7124820 | 7.101 | 1   | 1000 | 400 |
| 130 | -556  | 1447  | 1053  | 939132  | 6735449 | 7.172 | 1   | 1000 | 400 |
| 140 | -1270 | 1488  | 697   | 1052742 | 7617127 | 7.236 | 57  | 1000 | 400 |
| 150 | 653   | 1465  | 988   | 786670  | 5735463 | 7.291 | 48  | 1000 | 400 |
| 160 | 17189 | 18962 | 17660 | 137513  | 1006744 | 7.321 | 266 | 1000 | 400 |
| 170 | 35957 | 38487 | 36477 | 114504  | 833097  | 7.276 | 222 | 1000 | 400 |
| 180 | 51893 | 53648 | 52433 | 133314  | 966160  | 7.247 | 205 | 1000 | 400 |
| 190 | 66519 | 68308 | 67032 | 156245  | 1135965 | 7.27  | 280 | 1000 | 400 |

### B.3 Gossip Group Size Data

SD = The time interval between gossip messages.

TO = The timeout value for the Gossip protocol.

B = The number of nodes that each gossip message is sent to.

CD = The time interval between scans for failed nodes.

| nodes | min    | max    | avg    | msgs   | bytes    | bytes/msg | errors | SD  | TO  | B | CD  |
|-------|--------|--------|--------|--------|----------|-----------|--------|-----|-----|---|-----|
| 10    | 649    | 789    | 722    | 26672  | 1818808  | 68.192    | 0      | 100 | 450 | 4 | 450 |
| 20    | 407    | 918    | 607    | 53392  | 7290440  | 136.546   | 0      | 100 | 450 | 4 | 450 |
| 30    | 521    | 1050   | 594    | 74600  | 15255792 | 204.501   | 0      | 100 | 450 | 4 | 450 |
| 40    | 442    | 1022   | 695    | 98680  | 26906336 | 272.663   | 0      | 100 | 450 | 4 | 450 |
| 50    | 414    | 895    | 729    | 123560 | 42119724 | 340.885   | 0      | 100 | 450 | 4 | 450 |
| 60    | 489    | 823    | 667    | 143372 | 58596600 | 408.703   | 0      | 100 | 450 | 4 | 450 |
| 70    | 457    | 1056   | 802    | 158512 | 75470848 | 476.121   | 0      | 100 | 450 | 4 | 450 |
| 80    | 507    | 1127   | 848    | 161112 | 87351208 | 542.177   | 0      | 100 | 450 | 4 | 450 |
| 90    | 548    | 1025   | 851    | 156980 | 95263800 | 606.853   | 0      | 100 | 450 | 4 | 450 |
| 100   | 464    | 971    | 656    | 146344 | 98242560 | 671.313   | 0      | 100 | 450 | 4 | 450 |
| 110   | 482    | 1120   | 782    | 140204 | 1.04e+08 | 743.253   | 6      | 100 | 450 | 4 | 450 |
| 120   | 442    | 1089   | 740    | 126820 | 1.03e+08 | 811.119   | 23     | 100 | 450 | 4 | 450 |
| 130   | 415    | 1202   | 718    | 61248  | 50095960 | 817.92    | 0      | 100 | 450 | 4 | 450 |
| 140   | 559    | 1235   | 797    | 9532   | 7572525  | 794.432   | 3      | 100 | 450 | 4 | 450 |
| 150   | 18871  | 21296  | 19201  | 11363  | 9831334  | 865.206   | 21     | 100 | 450 | 4 | 450 |
| 160   | 35195  | 37304  | 35464  | 8849   | 7980885  | 901.897   | 18     | 100 | 450 | 4 | 450 |
| 170   | 47151  | 49635  | 47493  | 18367  | 18695948 | 1017.91   | 56     | 100 | 450 | 4 | 450 |
| 180   | 62721  | 64730  | 63152  | 15742  | 16827139 | 1068.933  | 9      | 100 | 450 | 4 | 450 |
| 190   | 78498  | 81419  | 78905  | 20308  | 23349196 | 1149.754  | 32     | 100 | 450 | 4 | 450 |
| 200   | 102489 | 103947 | 102796 | 13069  | 15279040 | 1169.106  | 673    | 100 | 450 | 4 | 450 |



## B.4 Random Group Size Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| nodes | min   | max    | avg    | msgs   | bytes  | bytes/msg | errors | SD | K | TO1 | TO2 |
|-------|-------|--------|--------|--------|--------|-----------|--------|----|---|-----|-----|
| 10    | 190   | 2810   | 1109   | 17941  | 112256 | 6.257     | 0      | 70 | 4 | 20  | 30  |
| 20    | 90    | 7800   | 1987   | 35411  | 234202 | 6.614     | 0      | 70 | 4 | 20  | 30  |
| 30    | 98    | 7577   | 2282   | 53272  | 362289 | 6.801     | 0      | 70 | 4 | 20  | 30  |
| 40    | 214   | 9214   | 2722   | 68929  | 472625 | 6.857     | 0      | 70 | 4 | 20  | 30  |
| 50    | 166   | 37307  | 5227   | 88667  | 613399 | 6.918     | 0      | 70 | 4 | 20  | 30  |
| 60    | 180   | 23409  | 4263   | 105419 | 731907 | 6.943     | 0      | 70 | 4 | 20  | 30  |
| 70    | 270   | 27868  | 5869   | 123307 | 860347 | 6.977     | 0      | 70 | 4 | 20  | 30  |
| 80    | 127   | 22347  | 4755   | 126479 | 884094 | 6.99      | 0      | 70 | 4 | 20  | 30  |
| 90    | 78    | 30449  | 7885   | 128536 | 902793 | 7.024     | 0      | 70 | 4 | 20  | 30  |
| 100   | 150   | 36844  | 7893   | 134874 | 959666 | 7.115     | 0      | 70 | 4 | 20  | 30  |
| 110   | 146   | 26756  | 6602   | 124188 | 890092 | 7.167     | 0      | 70 | 4 | 20  | 30  |
| 120   | 169   | 35893  | 10538  | 104526 | 759558 | 7.267     | 0      | 70 | 4 | 20  | 30  |
| 130   | 329   | 43964  | 9393   | 81723  | 601253 | 7.357     | 0      | 70 | 4 | 20  | 30  |
| 140   | 75    | 68043  | 11919  | 73987  | 553961 | 7.487     | 0      | 70 | 4 | 20  | 30  |
| 150   | 6501  | 91729  | 19006  | 57751  | 434570 | 7.525     | 0      | 70 | 4 | 20  | 30  |
| 160   | 15726 | 106866 | 28464  | 60425  | 454340 | 7.519     | 0      | 70 | 4 | 20  | 30  |
| 170   | 35020 | 100654 | 48790  | 68688  | 521516 | 7.593     | 0      | 70 | 4 | 20  | 30  |
| 180   | 42881 | 138127 | 58051  | 81519  | 626905 | 7.69      | 0      | 70 | 4 | 20  | 30  |
| 190   | 76865 | 162041 | 92275  | 86401  | 662174 | 7.664     | 0      | 70 | 4 | 20  | 30  |
| 200   | 88053 | 158163 | 104804 | 99553  | 771845 | 7.753     | 0      | 70 | 4 | 20  | 30  |

## B.5 Aggressive Group Size Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| nodes | min | max | avg | msgs  | bytes  | bytes/msg | errors | SD  | K | TO1 | TO2 |
|-------|-----|-----|-----|-------|--------|-----------|--------|-----|---|-----|-----|
| 10    | 329 | 332 | 331 | 12916 | 80807  | 6.256     | 0      | 100 | 4 | 20  | 60  |
| 20    | 140 | 142 | 141 | 25199 | 166353 | 6.602     | 0      | 100 | 4 | 20  | 60  |
| 30    | 115 | 196 | 120 | 37802 | 256598 | 6.788     | 1      | 100 | 4 | 20  | 60  |

|     |       |       |       |       |        |       |      |     |   |    |    |
|-----|-------|-------|-------|-------|--------|-------|------|-----|---|----|----|
| 40  | 251   | 255   | 253   | 49835 | 341446 | 6.852 | 0    | 100 | 4 | 20 | 60 |
| 50  | 105   | 119   | 110   | 63672 | 451784 | 7.095 | 0    | 100 | 4 | 20 | 60 |
| 60  | 110   | 119   | 114   | 73994 | 518820 | 7.012 | 1    | 100 | 4 | 20 | 60 |
| 70  | 117   | 127   | 124   | 83875 | 589200 | 7.025 | 0    | 100 | 4 | 20 | 60 |
| 80  | 117   | 130   | 121   | 84479 | 588160 | 6.962 | 0    | 100 | 4 | 20 | 60 |
| 90  | 297   | 310   | 301   | 88302 | 625738 | 7.086 | 0    | 100 | 4 | 20 | 60 |
| 100 | 515   | 525   | 522   | 88756 | 629914 | 7.097 | 186  | 100 | 4 | 20 | 60 |
| 110 | 528   | 540   | 535   | 97934 | 752733 | 7.686 | 847  | 100 | 4 | 20 | 60 |
| 120 | 109   | 120   | 113   | 64783 | 484604 | 7.48  | 285  | 100 | 4 | 20 | 60 |
| 130 | 220   | 233   | 224   | 52585 | 414079 | 7.874 | 552  | 100 | 4 | 20 | 60 |
| 140 | 181   | 254   | 200   | 17078 | 146303 | 8.567 | 517  | 100 | 4 | 20 | 60 |
| 150 | 4701  | 4787  | 4735  | 2028  | 17038  | 8.401 | 361  | 100 | 4 | 20 | 60 |
| 160 | 24349 | 24515 | 24382 | 3021  | 27836  | 9.214 | 0    | 100 | 4 | 20 | 60 |
| 170 | 41061 | 41331 | 41099 | 4038  | 37612  | 9.315 | 1340 | 100 | 4 | 20 | 60 |
| 180 | 59140 | 59294 | 59184 | 2630  | 21911  | 8.331 | 277  | 100 | 4 | 20 | 60 |
| 190 | 63930 | 64084 | 63966 | 10970 | 101804 | 9.28  | 845  | 100 | 4 | 20 | 60 |
| 200 | 92348 | 92420 | 92376 | 5087  | 41612  | 8.18  | 181  | 100 | 4 | 20 | 60 |

## C Network Reliability Data

drop% = The percentage of packets dropped in the network.

min = The minimum time for a node to detect the failure.

max = The maximum time for a node to detect the failure.

avg = The average time for a node to detect the failure.

msgs = The total number of messages sent by the nodes.

bytes = The total number of bytes sent by the nodes.

bytes/msg = The average number of bytes per message sent.

errors = The total number of false detections.

### C.1 Heartbeat Network Reliability Data

SD = The time interval between heartbeat messages.

TO = The Heartbeat timeout value.

CD = The time interval between scans for failed nodes.

| drop% | min  | max  | avg  | msgs   | bytes   | bytes/msg | errors | SD  | TO  | CD  |
|-------|------|------|------|--------|---------|-----------|--------|-----|-----|-----|
| 0     | 315  | 1046 | 600  | 708663 | 3494478 | 4.931     | 0      | 650 | 700 | 700 |
| 0.02  | -435 | 650  | 434  | 650939 | 3211927 | 4.934     | 1      | 650 | 700 | 700 |
| 0.04  | 39   | 1506 | 1078 | 612349 | 3021093 | 4.934     | 42     | 650 | 700 | 700 |

|      |        |      |       |        |         |       |     |     |     |     |
|------|--------|------|-------|--------|---------|-------|-----|-----|-----|-----|
| 0.06 | -857   | 1003 | 731   | 569487 | 2810042 | 4.934 | 22  | 650 | 700 | 700 |
| 0.08 | -588   | 991  | 630   | 516816 | 2548003 | 4.93  | 54  | 650 | 700 | 700 |
| 0.1  | -24    | 1276 | 826   | 601887 | 2968196 | 4.931 | 60  | 650 | 700 | 700 |
| 0.12 | -104   | 1257 | 844   | 508464 | 2508262 | 4.933 | 67  | 650 | 700 | 700 |
| 0.14 | -1044  | 841  | 386   | 526835 | 2598667 | 4.933 | 54  | 650 | 700 | 700 |
| 0.16 | -1829  | 882  | 233   | 536192 | 2646173 | 4.935 | 139 | 650 | 700 | 700 |
| 0.18 | -617   | 1312 | 750   | 543074 | 2678058 | 4.931 | 56  | 650 | 700 | 700 |
| 0.2  | -1005  | 1269 | 888   | 461433 | 2276246 | 4.933 | 71  | 650 | 700 | 700 |
| 0.22 | -1064  | 1090 | 569   | 483370 | 2383407 | 4.931 | 87  | 650 | 700 | 700 |
| 0.24 | -377   | 1273 | 897   | 463080 | 2283141 | 4.93  | 108 | 650 | 700 | 700 |
| 0.26 | -663   | 1312 | 665   | 467412 | 2305585 | 4.933 | 188 | 650 | 700 | 700 |
| 0.28 | -3519  | 1381 | 620   | 461371 | 2274169 | 4.929 | 194 | 650 | 700 | 700 |
| 0.3  | -1988  | 933  | 180   | 465649 | 2297916 | 4.935 | 182 | 650 | 700 | 700 |
| 0.32 | -4440  | 1254 | 608   | 410783 | 2026108 | 4.932 | 132 | 650 | 700 | 700 |
| 0.34 | -2253  | 1148 | 314   | 423006 | 2085706 | 4.931 | 224 | 650 | 700 | 700 |
| 0.36 | -1638  | 1273 | 707   | 402377 | 1983860 | 4.93  | 152 | 650 | 700 | 700 |
| 0.38 | -2519  | 1397 | 434   | 416856 | 2055988 | 4.932 | 262 | 650 | 700 | 700 |
| 0.4  | -2056  | 809  | 50    | 320246 | 1579312 | 4.932 | 297 | 650 | 700 | 700 |
| 0.42 | -2811  | 752  | 104   | 345798 | 1705592 | 4.932 | 326 | 650 | 700 | 700 |
| 0.44 | -3318  | 1184 | 391   | 327691 | 1617442 | 4.936 | 270 | 650 | 700 | 700 |
| 0.46 | -3915  | 1065 | 235   | 336322 | 1658798 | 4.932 | 272 | 650 | 700 | 700 |
| 0.48 | -1937  | 1076 | 156   | 323033 | 1592423 | 4.93  | 304 | 650 | 700 | 700 |
| 0.5  | -3383  | 1053 | 370   | 314834 | 1552566 | 4.931 | 360 | 650 | 700 | 700 |
| 0.52 | -5271  | 862  | -333  | 295316 | 1456679 | 4.933 | 378 | 650 | 700 | 700 |
| 0.54 | -6320  | 943  | -85   | 274503 | 1354186 | 4.933 | 413 | 650 | 700 | 700 |
| 0.56 | -8403  | 969  | -329  | 294702 | 1453087 | 4.931 | 395 | 650 | 700 | 700 |
| 0.58 | -5874  | 678  | -362  | 245049 | 1208964 | 4.934 | 349 | 650 | 700 | 700 |
| 0.6  | -5205  | 1341 | 101   | 255572 | 1260437 | 4.932 | 366 | 650 | 700 | 700 |
| 0.62 | -5289  | 862  | -374  | 246733 | 1216581 | 4.931 | 430 | 650 | 700 | 700 |
| 0.64 | -4914  | 1093 | -189  | 202468 | 998919  | 4.934 | 454 | 650 | 700 | 700 |
| 0.66 | -8115  | 1017 | -389  | 203580 | 1003956 | 4.932 | 473 | 650 | 700 | 700 |
| 0.68 | -6027  | 636  | -909  | 210634 | 1038601 | 4.931 | 461 | 650 | 700 | 700 |
| 0.7  | -6555  | 909  | -926  | 182106 | 897684  | 4.929 | 568 | 650 | 700 | 700 |
| 0.72 | -12210 | 556  | -1233 | 164864 | 813014  | 4.931 | 480 | 650 | 700 | 700 |
| 0.76 | -13711 | 701  | -1557 | 139254 | 686705  | 4.931 | 562 | 650 | 700 | 700 |
| 0.78 | -13647 | 8803 | -247  | 131671 | 649041  | 4.929 | 508 | 650 | 700 | 700 |
| 0.8  | -10977 | 880  | -1760 | 125084 | 617057  | 4.933 | 623 | 650 | 700 | 700 |

|      |        |      |        |        |        |       |     |     |     |     |
|------|--------|------|--------|--------|--------|-------|-----|-----|-----|-----|
| 0.82 | -9401  | 1245 | -1379  | 106557 | 525457 | 4.931 | 605 | 650 | 700 | 700 |
| 0.84 | -22920 | 829  | -3872  | 103418 | 509913 | 4.931 | 704 | 650 | 700 | 700 |
| 0.86 | -20693 | 701  | -3777  | 74033  | 365094 | 4.932 | 713 | 650 | 700 | 700 |
| 0.88 | -22372 | 1156 | -4531  | 64033  | 315794 | 4.932 | 713 | 650 | 700 | 700 |
| 0.9  | -29270 | 1240 | -5099  | 62181  | 306514 | 4.929 | 727 | 650 | 700 | 700 |
| 0.92 | -34466 | 840  | -5798  | 43362  | 213814 | 4.931 | 820 | 650 | 700 | 700 |
| 0.94 | -36911 | 839  | -8484  | 35546  | 175287 | 4.931 | 813 | 650 | 700 | 700 |
| 0.96 | -37974 | 498  | -12793 | 23693  | 116854 | 4.932 | 828 | 650 | 700 | 700 |
| 0.98 | -36280 | 671  | -22595 | 11445  | 56443  | 4.932 | 867 | 650 | 700 | 700 |
| 1    | -36211 | 0    | -36068 | 0      | 0      | ?     | 891 | 650 | 700 | 700 |

## C.2 Ping Network Reliability Data

SD = The time interval between ping messages.

ACK\_TO = The timeout value for a ping acknowledgment message.

| drop% | min    | max  | avg   | msgs   | bytes   | bytes/msg | errors | SD   | ACK_TO |
|-------|--------|------|-------|--------|---------|-----------|--------|------|--------|
| 0     | 413    | 1583 | 740   | 859768 | 5961000 | 6.933     | 2      | 1000 | 400    |
| 0.02  | -1378  | 1399 | 710   | 833448 | 5775571 | 6.93      | 23     | 1000 | 400    |
| 0.04  | -2466  | 1494 | 882   | 781845 | 5419849 | 6.932     | 34     | 1000 | 400    |
| 0.06  | -2061  | 1322 | 836   | 757120 | 5247720 | 6.931     | 65     | 1000 | 400    |
| 0.08  | -789   | 1489 | 847   | 699166 | 4846678 | 6.932     | 27     | 1000 | 400    |
| 0.1   | -1420  | 1460 | 844   | 709654 | 4919121 | 6.932     | 68     | 1000 | 400    |
| 0.12  | -2450  | 1409 | 437   | 665274 | 4610419 | 6.93      | 187    | 1000 | 400    |
| 0.14  | -3470  | 1397 | 452   | 645851 | 4475723 | 6.93      | 185    | 1000 | 400    |
| 0.16  | -3380  | 1406 | 547   | 604640 | 4190487 | 6.931     | 126    | 1000 | 400    |
| 0.18  | -1911  | 1356 | 692   | 617212 | 4278829 | 6.933     | 149    | 1000 | 400    |
| 0.2   | -3931  | 1383 | 598   | 607593 | 4211513 | 6.931     | 166    | 1000 | 400    |
| 0.22  | -6798  | 1491 | 252   | 582437 | 4037320 | 6.932     | 178    | 1000 | 400    |
| 0.24  | -3428  | 1559 | 261   | 504924 | 3498466 | 6.929     | 224    | 1000 | 400    |
| 0.26  | -5316  | 1419 | -34   | 521052 | 3611177 | 6.931     | 340    | 1000 | 400    |
| 0.28  | -4311  | 1277 | 1     | 528112 | 3659465 | 6.929     | 413    | 1000 | 400    |
| 0.3   | -4664  | 1408 | -149  | 491705 | 3407871 | 6.931     | 313    | 1000 | 400    |
| 0.32  | -7581  | 1065 | -566  | 498843 | 3456754 | 6.93      | 455    | 1000 | 400    |
| 0.34  | -8682  | 1589 | -145  | 443371 | 3073103 | 6.931     | 357    | 1000 | 400    |
| 0.36  | -15726 | 1431 | -657  | 431717 | 2992028 | 6.931     | 339    | 1000 | 400    |
| 0.38  | -7742  | 1589 | -714  | 413268 | 2864370 | 6.931     | 376    | 1000 | 400    |
| 0.4   | -11543 | 1387 | -1439 | 377440 | 2616888 | 6.933     | 500    | 1000 | 400    |

|      |        |      |        |        |         |       |     |      |     |
|------|--------|------|--------|--------|---------|-------|-----|------|-----|
| 0.42 | -10916 | 1390 | -1048  | 339826 | 2356067 | 6.933 | 451 | 1000 | 400 |
| 0.44 | -10032 | 1369 | -1043  | 347731 | 2409567 | 6.929 | 530 | 1000 | 400 |
| 0.46 | -20896 | 1154 | -1999  | 336692 | 2334048 | 6.932 | 560 | 1000 | 400 |
| 0.48 | -20654 | 1332 | -2053  | 326116 | 2260098 | 6.93  | 603 | 1000 | 400 |
| 0.5  | -21105 | 1349 | -2132  | 311744 | 2160987 | 6.932 | 584 | 1000 | 400 |
| 0.52 | -15892 | 1393 | -2334  | 266645 | 1847983 | 6.93  | 585 | 1000 | 400 |
| 0.54 | -21743 | 1390 | -3400  | 246053 | 1705486 | 6.931 | 581 | 1000 | 400 |
| 0.56 | -22212 | 1236 | -3511  | 256909 | 1781187 | 6.933 | 641 | 1000 | 400 |
| 0.58 | -25167 | 1265 | -3569  | 230054 | 1594460 | 6.931 | 674 | 1000 | 400 |
| 0.6  | -27322 | 1286 | -3809  | 214575 | 1487071 | 6.93  | 741 | 1000 | 400 |
| 0.62 | -34061 | 1410 | -4882  | 215882 | 1496405 | 6.932 | 714 | 1000 | 400 |
| 0.64 | -40877 | 1416 | -6645  | 207279 | 1436827 | 6.932 | 719 | 1000 | 400 |
| 0.66 | -37233 | 1197 | -5736  | 174450 | 1209345 | 6.932 | 792 | 1000 | 400 |
| 0.68 | -41979 | 1460 | -6616  | 183867 | 1273821 | 6.928 | 741 | 1000 | 400 |
| 0.7  | -41112 | 1246 | -9006  | 165033 | 1143985 | 6.932 | 776 | 1000 | 400 |
| 0.72 | -40072 | 1328 | -10728 | 147261 | 1020982 | 6.933 | 773 | 1000 | 400 |
| 0.74 | -37871 | 666  | -11219 | 127460 | 883101  | 6.928 | 810 | 1000 | 400 |
| 0.76 | -41964 | 1391 | -12987 | 127856 | 885968  | 6.929 | 812 | 1000 | 400 |
| 0.78 | -39493 | 1016 | -18377 | 107147 | 742356  | 6.928 | 840 | 1000 | 400 |
| 0.8  | -37020 | 1396 | -18962 | 91749  | 635767  | 6.929 | 826 | 1000 | 400 |
| 0.82 | -38560 | 1391 | -19421 | 82397  | 570959  | 6.929 | 853 | 1000 | 400 |
| 0.86 | -37470 | 1115 | -25176 | 59964  | 415628  | 6.931 | 864 | 1000 | 400 |
| 0.88 | -38951 | 677  | -29622 | 53282  | 369076  | 6.927 | 871 | 1000 | 400 |
| 0.9  | -38991 | 949  | -32693 | 44131  | 305792  | 6.929 | 873 | 1000 | 400 |
| 0.92 | -34592 | 0    | -31486 | 30268  | 209714  | 6.929 | 883 | 1000 | 400 |
| 0.94 | -40823 | 623  | -38189 | 26191  | 181536  | 6.931 | 888 | 1000 | 400 |
| 0.96 | -37880 | 0    | -36724 | 16054  | 111284  | 6.932 | 889 | 1000 | 400 |
| 0.98 | -35609 | 0    | -34938 | 7352   | 51009   | 6.938 | 889 | 1000 | 400 |
| 1    | -35862 | 0    | -35471 | 0      | 0       | ?     | 891 | 1000 | 400 |

### C.3 Gossip Network Reliability Data

SD = The time interval between gossip messages.

TO = The timeout value for the Gossip protocol.

B = The number of nodes that each gossip message is sent to.

CD = The time interval between scans for failed nodes.

| drop% | min | max  | avg | msgs   | bytes | bytes/msg | errors | SD  | TO  | B | CD  |
|-------|-----|------|-----|--------|-------|-----------|--------|-----|-----|---|-----|
| 0     | 522 | 1235 | 759 | 148956 | 1e+08 | 671.882   | 0      | 100 | 450 | 4 | 450 |

|      |      |      |      |        |          |         |    |     |     |   |     |
|------|------|------|------|--------|----------|---------|----|-----|-----|---|-----|
| 0.02 | 427  | 1060 | 813  | 147103 | 98860612 | 672.05  | 0  | 100 | 450 | 4 | 450 |
| 0.04 | 581  | 1147 | 757  | 146688 | 98661521 | 672.594 | 3  | 100 | 450 | 4 | 450 |
| 0.06 | 438  | 993  | 652  | 147633 | 99412376 | 673.375 | 0  | 100 | 450 | 4 | 450 |
| 0.08 | 497  | 1057 | 789  | 150501 | 1.02e+08 | 674.536 | 0  | 100 | 450 | 4 | 450 |
| 0.1  | 479  | 965  | 835  | 141518 | 95298046 | 673.399 | 0  | 100 | 450 | 4 | 450 |
| 0.12 | 485  | 1150 | 823  | 143899 | 97040405 | 674.365 | 0  | 100 | 450 | 4 | 450 |
| 0.14 | 456  | 1150 | 763  | 126938 | 85252126 | 671.604 | 0  | 100 | 450 | 4 | 450 |
| 0.16 | 568  | 1163 | 811  | 122232 | 82031277 | 671.111 | 2  | 100 | 450 | 4 | 450 |
| 0.18 | 419  | 1118 | 687  | 128083 | 86214750 | 673.116 | 0  | 100 | 450 | 4 | 450 |
| 0.2  | 486  | 1210 | 923  | 129148 | 87046616 | 674.007 | 0  | 100 | 450 | 4 | 450 |
| 0.22 | 378  | 1121 | 758  | 125265 | 84426026 | 673.979 | 2  | 100 | 450 | 4 | 450 |
| 0.24 | 607  | 1118 | 761  | 105732 | 70801681 | 669.633 | 2  | 100 | 450 | 4 | 450 |
| 0.26 | 423  | 1069 | 807  | 110231 | 74029310 | 671.583 | 0  | 100 | 450 | 4 | 450 |
| 0.28 | 391  | 1074 | 736  | 107942 | 72500182 | 671.659 | 1  | 100 | 450 | 4 | 450 |
| 0.3  | 431  | 1072 | 805  | 107482 | 72279755 | 672.482 | 16 | 100 | 450 | 4 | 450 |
| 0.32 | 492  | 1094 | 850  | 104360 | 70184314 | 672.521 | 0  | 100 | 450 | 4 | 450 |
| 0.34 | 442  | 1254 | 797  | 86698  | 57863516 | 667.415 | 0  | 100 | 450 | 4 | 450 |
| 0.36 | 541  | 1159 | 875  | 98255  | 66065773 | 672.391 | 1  | 100 | 450 | 4 | 450 |
| 0.38 | 142  | 1166 | 899  | 90758  | 60892303 | 670.93  | 3  | 100 | 450 | 4 | 450 |
| 0.4  | 451  | 1231 | 794  | 91335  | 61392360 | 672.167 | 1  | 100 | 450 | 4 | 450 |
| 0.42 | 427  | 1136 | 777  | 88128  | 59225340 | 672.038 | 0  | 100 | 450 | 4 | 450 |
| 0.44 | 454  | 1249 | 800  | 89632  | 60366446 | 673.492 | 20 | 100 | 450 | 4 | 450 |
| 0.46 | 153  | 1304 | 980  | 80720  | 54207756 | 671.553 | 0  | 100 | 450 | 4 | 450 |
| 0.48 | 638  | 1388 | 1047 | 85636  | 57746346 | 674.323 | 2  | 100 | 450 | 4 | 450 |
| 0.5  | 257  | 1047 | 649  | 78915  | 53094501 | 672.806 | 2  | 100 | 450 | 4 | 450 |
| 0.52 | 501  | 1420 | 904  | 73640  | 49501808 | 672.214 | 3  | 100 | 450 | 4 | 450 |
| 0.54 | 446  | 1361 | 916  | 70655  | 47498323 | 672.257 | 2  | 100 | 450 | 4 | 450 |
| 0.56 | 516  | 1309 | 789  | 73776  | 49768657 | 674.591 | 2  | 100 | 450 | 4 | 450 |
| 0.58 | 677  | 1373 | 1008 | 67738  | 45614160 | 673.391 | 29 | 100 | 450 | 4 | 450 |
| 0.6  | 613  | 1490 | 1055 | 59909  | 40197349 | 670.973 | 6  | 100 | 450 | 4 | 450 |
| 0.62 | 580  | 1492 | 988  | 62580  | 42167881 | 673.824 | 3  | 100 | 450 | 4 | 450 |
| 0.64 | 22   | 1586 | 969  | 58356  | 39301041 | 673.47  | 17 | 100 | 450 | 4 | 450 |
| 0.66 | 607  | 1703 | 1126 | 53679  | 36101372 | 672.542 | 2  | 100 | 450 | 4 | 450 |
| 0.68 | 227  | 1921 | 1406 | 49765  | 33435341 | 671.865 | 45 | 100 | 450 | 4 | 450 |
| 0.7  | 294  | 1679 | 1147 | 42803  | 28636056 | 669.02  | 39 | 100 | 450 | 4 | 450 |
| 0.72 | 14   | 1522 | 939  | 46534  | 31333712 | 673.351 | 31 | 100 | 450 | 4 | 450 |
| 0.74 | -330 | 1550 | 1046 | 39842  | 26733887 | 670.998 | 87 | 100 | 450 | 4 | 450 |

|      |        |      |        |       |          |         |     |     |     |   |     |
|------|--------|------|--------|-------|----------|---------|-----|-----|-----|---|-----|
| 0.76 | -53    | 1617 | 1040   | 32985 | 21992674 | 666.748 | 64  | 100 | 450 | 4 | 450 |
| 0.78 | -377   | 1939 | 933    | 36041 | 24236900 | 672.481 | 58  | 100 | 450 | 4 | 450 |
| 0.8  | -292   | 2082 | 1080   | 27991 | 18676592 | 667.236 | 129 | 100 | 450 | 4 | 450 |
| 0.82 | -501   | 1937 | 1219   | 28321 | 18983763 | 670.307 | 130 | 100 | 450 | 4 | 450 |
| 0.84 | -583   | 2811 | 1384   | 26416 | 17746501 | 671.809 | 185 | 100 | 450 | 4 | 450 |
| 0.86 | -349   | 2151 | 760    | 18910 | 12551664 | 663.758 | 208 | 100 | 450 | 4 | 450 |
| 0.88 | -501   | 2153 | 828    | 17642 | 11756664 | 666.402 | 326 | 100 | 450 | 4 | 450 |
| 0.9  | -2137  | 2100 | 664    | 15157 | 10088563 | 665.604 | 333 | 100 | 450 | 4 | 450 |
| 0.92 | -2435  | 3982 | 1691   | 12401 | 8248529  | 665.15  | 493 | 100 | 450 | 4 | 450 |
| 0.94 | -3130  | 3854 | -541   | 8533  | 5627263  | 659.471 | 538 | 100 | 450 | 4 | 450 |
| 0.96 | -3991  | 4309 | 3      | 5873  | 3844118  | 654.541 | 708 | 100 | 450 | 4 | 450 |
| 0.98 | -8487  | 3606 | -2577  | 2583  | 1628463  | 630.454 | 806 | 100 | 450 | 4 | 450 |
| 1    | -37876 | 0    | -37692 | 0     | 0        | ?       | 891 | 100 | 450 | 4 | 450 |

## C.4 Random Ping Network Reliability Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| drop% | min   | max   | avg  | msgs   | bytes   | bytes/msg | errors | SD | K | TO1 | TO2 |
|-------|-------|-------|------|--------|---------|-----------|--------|----|---|-----|-----|
| 0     | 173   | 50315 | 8147 | 127084 | 893653  | 7.032     | 0      | 70 | 4 | 20  | 30  |
| 0.02  | 227   | 34343 | 7609 | 159658 | 1228827 | 7.697     | 0      | 70 | 4 | 20  | 30  |
| 0.04  | 244   | 33223 | 7420 | 186913 | 1503064 | 8.042     | 1      | 70 | 4 | 20  | 30  |
| 0.06  | 114   | 50634 | 8892 | 208588 | 1734246 | 8.314     | 0      | 70 | 4 | 20  | 30  |
| 0.08  | 102   | 41096 | 6903 | 228566 | 1945179 | 8.51      | 0      | 70 | 4 | 20  | 30  |
| 0.1   | 70    | 64301 | 9511 | 251385 | 2172346 | 8.642     | 11     | 70 | 4 | 20  | 30  |
| 0.12  | 183   | 44937 | 7763 | 283671 | 2478980 | 8.739     | 0      | 70 | 4 | 20  | 30  |
| 0.14  | -1661 | 27921 | 6726 | 248748 | 2202149 | 8.853     | 0      | 70 | 4 | 20  | 30  |
| 0.16  | 50    | 44328 | 8152 | 286638 | 2549758 | 8.895     | 2      | 70 | 4 | 20  | 30  |
| 0.18  | 208   | 50590 | 8893 | 292325 | 2617873 | 8.955     | 2      | 70 | 4 | 20  | 30  |
| 0.2   | -3577 | 58574 | 8188 | 311576 | 2802438 | 8.994     | 1      | 70 | 4 | 20  | 30  |
| 0.22  | 180   | 38214 | 8261 | 310816 | 2812313 | 9.048     | 16     | 70 | 4 | 20  | 30  |
| 0.24  | -968  | 42029 | 6707 | 301427 | 2735414 | 9.075     | 29     | 70 | 4 | 20  | 30  |
| 0.26  | -3968 | 42043 | 7053 | 306158 | 2783519 | 9.092     | 19     | 70 | 4 | 20  | 30  |
| 0.28  | -4212 | 46281 | 7139 | 300556 | 2739897 | 9.116     | 24     | 70 | 4 | 20  | 30  |
| 0.3   | -2926 | 26659 | 7439 | 303075 | 2769357 | 9.138     | 40     | 70 | 4 | 20  | 30  |

|      |        |       |        |        |         |       |     |    |   |    |    |
|------|--------|-------|--------|--------|---------|-------|-----|----|---|----|----|
| 0.32 | -7190  | 41670 | 6203   | 270213 | 2474286 | 9.157 | 50  | 70 | 4 | 20 | 30 |
| 0.34 | -6384  | 38254 | 6322   | 286968 | 2631852 | 9.171 | 47  | 70 | 4 | 20 | 30 |
| 0.36 | -2126  | 26266 | 7181   | 291113 | 2674500 | 9.187 | 64  | 70 | 4 | 20 | 30 |
| 0.38 | -5351  | 35674 | 6332   | 255322 | 2351409 | 9.21  | 70  | 70 | 4 | 20 | 30 |
| 0.4  | -5348  | 38339 | 7155   | 268150 | 2469389 | 9.209 | 87  | 70 | 4 | 20 | 30 |
| 0.42 | -6293  | 29469 | 6208   | 233038 | 2146926 | 9.213 | 101 | 70 | 4 | 20 | 30 |
| 0.44 | -7484  | 32507 | 6117   | 252688 | 2334293 | 9.238 | 108 | 70 | 4 | 20 | 30 |
| 0.46 | -10041 | 33734 | 5264   | 248606 | 2295356 | 9.233 | 103 | 70 | 4 | 20 | 30 |
| 0.48 | -6439  | 34642 | 6760   | 250507 | 2315080 | 9.242 | 126 | 70 | 4 | 20 | 30 |
| 0.5  | -5431  | 34976 | 7134   | 216984 | 2003386 | 9.233 | 129 | 70 | 4 | 20 | 30 |
| 0.52 | -6830  | 33017 | 5020   | 201068 | 1861149 | 9.256 | 141 | 70 | 4 | 20 | 30 |
| 0.54 | -8484  | 28149 | 6027   | 194961 | 1805095 | 9.259 | 188 | 70 | 4 | 20 | 30 |
| 0.56 | -10923 | 32355 | 4570   | 164666 | 1525698 | 9.265 | 189 | 70 | 4 | 20 | 30 |
| 0.58 | -14778 | 34840 | 5144   | 169604 | 1571349 | 9.265 | 187 | 70 | 4 | 20 | 30 |
| 0.6  | -11504 | 25213 | 3575   | 164429 | 1522520 | 9.259 | 211 | 70 | 4 | 20 | 30 |
| 0.62 | -16519 | 36223 | 4701   | 175869 | 1630314 | 9.27  | 215 | 70 | 4 | 20 | 30 |
| 0.64 | -16676 | 29631 | 4224   | 145927 | 1352724 | 9.27  | 220 | 70 | 4 | 20 | 30 |
| 0.66 | -20204 | 38092 | 4439   | 142173 | 1317140 | 9.264 | 272 | 70 | 4 | 20 | 30 |
| 0.68 | -18056 | 48135 | 3596   | 124914 | 1158391 | 9.274 | 303 | 70 | 4 | 20 | 30 |
| 0.72 | -17517 | 39180 | 5178   | 110953 | 1027730 | 9.263 | 316 | 70 | 4 | 20 | 30 |
| 0.74 | -38027 | 37336 | 1316   | 95846  | 888278  | 9.268 | 318 | 70 | 4 | 20 | 30 |
| 0.76 | -19121 | 33252 | 3219   | 81873  | 758915  | 9.269 | 389 | 70 | 4 | 20 | 30 |
| 0.78 | -22269 | 31839 | 1576   | 70513  | 654113  | 9.276 | 399 | 70 | 4 | 20 | 30 |
| 0.8  | -32128 | 33638 | 2415   | 69315  | 643036  | 9.277 | 446 | 70 | 4 | 20 | 30 |
| 0.82 | -23295 | 26468 | 1008   | 61200  | 567559  | 9.274 | 451 | 70 | 4 | 20 | 30 |
| 0.84 | -34622 | 32380 | -2539  | 52239  | 485119  | 9.287 | 500 | 70 | 4 | 20 | 30 |
| 0.86 | -37350 | 33337 | -2341  | 46055  | 426765  | 9.266 | 530 | 70 | 4 | 20 | 30 |
| 0.88 | -33688 | 24057 | -2735  | 35022  | 325164  | 9.285 | 540 | 70 | 4 | 20 | 30 |
| 0.9  | -35620 | 23096 | -3896  | 32107  | 297359  | 9.262 | 563 | 70 | 4 | 20 | 30 |
| 0.92 | -35051 | 17770 | -6405  | 22203  | 206109  | 9.283 | 641 | 70 | 4 | 20 | 30 |
| 0.94 | -41080 | 24600 | -12773 | 18636  | 172664  | 9.265 | 666 | 70 | 4 | 20 | 30 |
| 0.96 | -37786 | 22709 | -15253 | 11181  | 103516  | 9.258 | 741 | 70 | 4 | 20 | 30 |
| 0.98 | -36800 | 20814 | -20074 | 5277   | 48994   | 9.284 | 811 | 70 | 4 | 20 | 30 |
| 1    | -31813 | 5059  | -24429 | 0      | 0       | ?     | 879 | 70 | 4 | 20 | 30 |



## C.5 Aggressive Network Reliability Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| drop% | min    | max   | avg   | msgs   | bytes   | bytes/msg | errors | SD  | K | TO1 | TO2 |
|-------|--------|-------|-------|--------|---------|-----------|--------|-----|---|-----|-----|
| 0     | 244    | 252   | 248   | 80173  | 579601  | 7.229     | 0      | 100 | 4 | 20  | 60  |
| 0.02  | 256    | 1811  | 307   | 105993 | 813444  | 7.675     | 131    | 100 | 4 | 20  | 60  |
| 0.04  | 178    | 215   | 184   | 130051 | 1054588 | 8.109     | 150    | 100 | 4 | 20  | 60  |
| 0.06  | 132    | 5073  | 284   | 148884 | 1223621 | 8.219     | 217    | 100 | 4 | 20  | 60  |
| 0.08  | -9300  | 4031  | -138  | 191067 | 1624945 | 8.505     | 30     | 100 | 4 | 20  | 60  |
| 0.1   | -2037  | 8352  | -1016 | 185179 | 1577060 | 8.516     | 44     | 100 | 4 | 20  | 60  |
| 0.12  | 148    | 3869  | 260   | 182228 | 1559060 | 8.556     | 71     | 100 | 4 | 20  | 60  |
| 0.14  | 133    | 7421  | 452   | 225170 | 1970069 | 8.749     | 112    | 100 | 4 | 20  | 60  |
| 0.16  | 126    | 3668  | 304   | 211382 | 1846337 | 8.735     | 0      | 100 | 4 | 20  | 60  |
| 0.18  | 187    | 2118  | 366   | 239333 | 2120925 | 8.862     | 70     | 100 | 4 | 20  | 60  |
| 0.2   | -5607  | 492   | -151  | 243538 | 2169037 | 8.906     | 61     | 100 | 4 | 20  | 60  |
| 0.22  | -3413  | 4900  | -205  | 265498 | 2372128 | 8.935     | 155    | 100 | 4 | 20  | 60  |
| 0.24  | -3826  | 7090  | 177   | 263757 | 2382299 | 9.032     | 293    | 100 | 4 | 20  | 60  |
| 0.26  | 602    | 8602  | 806   | 289291 | 2628094 | 9.085     | 211    | 100 | 4 | 20  | 60  |
| 0.28  | -8232  | 9191  | 165   | 259766 | 2375130 | 9.143     | 327    | 100 | 4 | 20  | 60  |
| 0.3   | -10565 | 5619  | -1175 | 293414 | 2697737 | 9.194     | 319    | 100 | 4 | 20  | 60  |
| 0.32  | -6276  | 5936  | -412  | 326030 | 3010524 | 9.234     | 436    | 100 | 4 | 20  | 60  |
| 0.34  | -6265  | 398   | -984  | 334337 | 3115406 | 9.318     | 360    | 100 | 4 | 20  | 60  |
| 0.36  | -2679  | 518   | -222  | 356467 | 3335889 | 9.358     | 434    | 100 | 4 | 20  | 60  |
| 0.38  | -3582  | 5530  | 366   | 377158 | 3543078 | 9.394     | 391    | 100 | 4 | 20  | 60  |
| 0.4   | -2450  | 9904  | -73   | 377480 | 3566809 | 9.449     | 464    | 100 | 4 | 20  | 60  |
| 0.42  | -4966  | 2006  | -41   | 411241 | 3895661 | 9.473     | 485    | 100 | 4 | 20  | 60  |
| 0.44  | -2404  | 5574  | 118   | 409861 | 3897451 | 9.509     | 386    | 100 | 4 | 20  | 60  |
| 0.46  | -5098  | 8172  | -924  | 439171 | 4195146 | 9.552     | 324    | 100 | 4 | 20  | 60  |
| 0.48  | -3197  | 8667  | 76    | 432556 | 4147694 | 9.589     | 517    | 100 | 4 | 20  | 60  |
| 0.5   | -5748  | 10289 | -720  | 455573 | 4379572 | 9.613     | 485    | 100 | 4 | 20  | 60  |
| 0.52  | -6667  | 2869  | -809  | 477937 | 4599865 | 9.624     | 475    | 100 | 4 | 20  | 60  |
| 0.54  | -2040  | 10206 | 138   | 465004 | 4489965 | 9.656     | 527    | 100 | 4 | 20  | 60  |
| 0.56  | -2927  | 10287 | 255   | 466958 | 4517556 | 9.674     | 528    | 100 | 4 | 20  | 60  |
| 0.58  | -2711  | 7073  | 34    | 459455 | 4454818 | 9.696     | 459    | 100 | 4 | 20  | 60  |
| 0.6   | -3378  | 4292  | -52   | 440993 | 4279481 | 9.704     | 500    | 100 | 4 | 20  | 60  |

|      |        |       |        |        |         |       |     |     |   |    |    |
|------|--------|-------|--------|--------|---------|-------|-----|-----|---|----|----|
| 0.62 | -1666  | 8272  | 862    | 530036 | 5148596 | 9.714 | 460 | 100 | 4 | 20 | 60 |
| 0.64 | -4054  | 6965  | -278   | 460673 | 4483030 | 9.731 | 440 | 100 | 4 | 20 | 60 |
| 0.66 | -2497  | 7584  | -291   | 440199 | 4287461 | 9.74  | 546 | 100 | 4 | 20 | 60 |
| 0.68 | -2815  | 4449  | -98    | 490777 | 4783293 | 9.746 | 496 | 100 | 4 | 20 | 60 |
| 0.7  | -4158  | 9959  | 109    | 446833 | 4356649 | 9.75  | 442 | 100 | 4 | 20 | 60 |
| 0.72 | -2253  | 6501  | 218    | 454550 | 4434082 | 9.755 | 506 | 100 | 4 | 20 | 60 |
| 0.74 | -4989  | 9538  | 312    | 404949 | 3954534 | 9.766 | 434 | 100 | 4 | 20 | 60 |
| 0.76 | -3789  | 10660 | 584    | 411298 | 4017939 | 9.769 | 475 | 100 | 4 | 20 | 60 |
| 0.78 | -3274  | 10651 | 422    | 354623 | 3471278 | 9.789 | 507 | 100 | 4 | 20 | 60 |
| 0.8  | -2566  | 11049 | 281    | 285232 | 2792949 | 9.792 | 556 | 100 | 4 | 20 | 60 |
| 0.82 | -15099 | 9172  | 492    | 331555 | 3245855 | 9.79  | 486 | 100 | 4 | 20 | 60 |
| 0.84 | -9360  | 10471 | 550    | 279533 | 2738256 | 9.796 | 397 | 100 | 4 | 20 | 60 |
| 0.86 | -5817  | 9360  | 320    | 241842 | 2373512 | 9.814 | 526 | 100 | 4 | 20 | 60 |
| 0.88 | -9447  | 8755  | 86     | 195878 | 1921118 | 9.808 | 484 | 100 | 4 | 20 | 60 |
| 0.9  | -9949  | 10335 | -207   | 172664 | 1693220 | 9.806 | 499 | 100 | 4 | 20 | 60 |
| 0.92 | -8407  | 9019  | -94    | 149149 | 1462683 | 9.807 | 546 | 100 | 4 | 20 | 60 |
| 0.94 | -15630 | 9344  | -563   | 109974 | 1078935 | 9.811 | 580 | 100 | 4 | 20 | 60 |
| 0.96 | -19962 | 9038  | -1395  | 81896  | 804416  | 9.822 | 605 | 100 | 4 | 20 | 60 |
| 0.98 | -37721 | 6247  | -7451  | 38495  | 378063  | 9.821 | 672 | 100 | 4 | 20 | 60 |
| 1    | -39220 | 0     | -33562 | 0      | 0       | ?     | 891 | 100 | 4 | 20 | 60 |

## D Device Type Data

type = The device type tested.

0 = Default

1 = Lossy

2 = Slow

3 = SlowLossy

4 = XtraLossy

5 = XtraSlow

6 = XtraSlowXtraLossy

7 = Worst

min = The minimum time for a node to detect the failure.

max = The maximum time for a node to detect the failure.

avg = The average time for a node to detect the failure.

msgs = The total number of messages sent by the nodes.

bytes = The total number of bytes sent by the nodes.

bytes/msg = The average number of bytes per message sent.

errors = The total number of false detections.

## D.1 Heartbeat Device Type Data

SD = The time interval between heartbeat messages.

TO = The Heartbeat timeout value.

CD = The time interval between scans for failed nodes.

| type | min | max  | avg  | msgs   | bytes   | bytes/msg | errors | SD  | TO  | CD  |
|------|-----|------|------|--------|---------|-----------|--------|-----|-----|-----|
| 0    | 145 | 739  | 519  | 627800 | 3098600 | 4.936     | 0      | 650 | 700 | 700 |
| 1    | 286 | 980  | 574  | 631849 | 3115581 | 4.931     | 0      | 650 | 700 | 700 |
| 2    | 155 | 1040 | 564  | 625678 | 3086190 | 4.933     | 4      | 650 | 700 | 700 |
| 3    | 111 | 1085 | 436  | 650256 | 3206145 | 4.931     | 0      | 650 | 700 | 700 |
| 4    | 160 | 854  | 599  | 626735 | 3089538 | 4.93      | 0      | 650 | 700 | 700 |
| 5    | 180 | 1560 | 680  | 575321 | 2836444 | 4.93      | 0      | 650 | 700 | 700 |
| 6    | 573 | 1912 | 984  | 659901 | 3252650 | 4.929     | 0      | 650 | 700 | 700 |
| 7    | 447 | 4220 | 1497 | 403632 | 1984833 | 4.917     | 0      | 650 | 700 | 700 |

## D.2 Ping Device Type Data

SD = The time interval between ping messages.

ACK\_TO = The timeout value for a ping acknowledgment message.

| type | min    | max  | avg  | msgs   | bytes   | bytes/msg | errors | SD   | ACK_TO |
|------|--------|------|------|--------|---------|-----------|--------|------|--------|
| 0    | 417    | 1378 | 716  | 839649 | 5818466 | 6.93      | 18     | 1000 | 400    |
| 1    | -182   | 1376 | 883  | 823909 | 5709647 | 6.93      | 48     | 1000 | 400    |
| 2    | 461    | 1499 | 977  | 814907 | 5648009 | 6.931     | 0      | 1000 | 400    |
| 3    | -451   | 1776 | 830  | 366837 | 2541971 | 6.929     | 53     | 1000 | 400    |
| 4    | 590    | 1379 | 920  | 605812 | 4197944 | 6.929     | 85     | 1000 | 400    |
| 5    | 399    | 2599 | 1238 | 646527 | 4479185 | 6.928     | 0      | 1000 | 400    |
| 6    | -2681  | 2619 | 1079 | 651011 | 4509774 | 6.927     | 88     | 1000 | 400    |
| 7    | -28689 | 9104 | -624 | 415760 | 2874302 | 6.913     | 247    | 1000 | 400    |

## D.3 Gossip Device Type Data

SD = The time interval between gossip messages.

TO = The timeout value for the Gossip protocol.

B = The number of nodes that each gossip message is sent to.

CD = The time interval between scans for failed nodes.

| type | min | max | avg | msgs | bytes | bytes/msg | errors | SD | TO | B | CD |
|------|-----|-----|-----|------|-------|-----------|--------|----|----|---|----|
|------|-----|-----|-----|------|-------|-----------|--------|----|----|---|----|

|   |     |      |     |        |          |         |   |     |     |   |     |
|---|-----|------|-----|--------|----------|---------|---|-----|-----|---|-----|
| 0 | 494 | 1045 | 741 | 156440 | 1.05e+08 | 673.319 | 0 | 100 | 450 | 4 | 450 |
| 1 | 436 | 1102 | 762 | 148046 | 99553198 | 672.448 | 0 | 100 | 450 | 4 | 450 |
| 2 | 575 | 1128 | 773 | 145923 | 97947874 | 671.23  | 4 | 100 | 450 | 4 | 450 |
| 3 | 436 | 1025 | 726 | 158104 | 1.07e+08 | 674.333 | 1 | 100 | 450 | 4 | 450 |
| 4 | 352 | 978  | 689 | 154495 | 1.04e+08 | 674.416 | 0 | 100 | 450 | 4 | 450 |
| 5 | 407 | 1163 | 711 | 154562 | 1.04e+08 | 672.869 | 0 | 100 | 450 | 4 | 450 |
| 6 | 434 | 1160 | 831 | 147575 | 99248473 | 672.529 | 1 | 100 | 450 | 4 | 450 |
| 7 | 498 | 1519 | 929 | 122395 | 81704554 | 667.548 | 7 | 100 | 450 | 4 | 450 |

## D.4 Random Ping Device Type Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| type | min       | max   | avg    | msgs    | bytes   | bytes/msg | errors | SD | K  | TO1 | TO2 |
|------|-----------|-------|--------|---------|---------|-----------|--------|----|----|-----|-----|
| 0    | 106       | 51319 | 9167   | 129009  | 906884  | 7.03      | 0      | 70 | 4  | 20  | 30  |
| 1    | 167       | 57754 | 9840   | 100772  | 820367  | 8.141     | 2      | 70 | 4  | 20  | 30  |
| 2    | 166       | 34550 | 7823   | 162936  | 1234732 | 7.578     | 0      | 70 | 4  | 20  | 30  |
| 3    | 81        | 45686 | 7708   | 194565  | 1574634 | 8.093     | 2      | 70 | 4  | 20  | 30  |
| 4    | 150       | 39865 | 7746   | 203208  | 1662488 | 8.181     | 0      | 70 | 4  | 20  | 30  |
| 5    | 217       | 43662 | 8961   | 278464  | 2418946 | 8.687     | 0      | 70 | 4  | 20  | 30  |
| 6    | 106       | 39098 | 9109   | 298818  | 2639874 | 8.834     | 1      | 70 | 4  | 20  | 30  |
| 7    | 149120017 | 18319 | 250841 | 2306778 | 9.196   | 0         | 70     | 4  | 20 | 30  |     |

## D.5 Aggressive Device Type Data

SD = The time interval between ping messages.

K = The number of nodes used in a swarm procedure.

TO1 = The timeout value for the acknowledgment of an initial ping message.

TO2 = The timeout value for the swarm procedure.

| type | min | max  | avg | msgs   | bytes   | bytes/msg | errors | SD  | K | TO1 | TO2 |
|------|-----|------|-----|--------|---------|-----------|--------|-----|---|-----|-----|
| 0    | 101 | 263  | 124 | 88150  | 702576  | 7.97      | 0      | 100 | 4 | 20  | 60  |
| 1    | 172 | 5969 | 293 | 108503 | 840160  | 7.743     | 11     | 100 | 4 | 20  | 60  |
| 2    | 100 | 131  | 110 | 118738 | 953983  | 8.034     | 0      | 100 | 4 | 20  | 60  |
| 3    | 308 | 6229 | 421 | 199382 | 1700215 | 8.527     | 0      | 100 | 4 | 20  | 60  |
| 4    | 93  | 302  | 116 | 155004 | 1327453 | 8.564     | 0      | 100 | 4 | 20  | 60  |
| 5    | 307 | 1429 | 474 | 374845 | 3423637 | 9.133     | 0      | 100 | 4 | 20  | 60  |

|   |     |       |      |        |         |       |   |     |   |    |    |
|---|-----|-------|------|--------|---------|-------|---|-----|---|----|----|
| 6 | 273 | 3056  | 408  | 392484 | 3592603 | 9.154 | 0 | 100 | 4 | 20 | 60 |
| 7 | 174 | 48032 | 2143 | 247970 | 2264059 | 9.13  | 0 | 100 | 4 | 20 | 60 |

## References

- [1] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Workshop on Distributed Algorithms*, pages 126–140, 1997.
- [2] Marcos Kawazoe Aguilera, Wei Chen, and Sam Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science*, 220(1):3–30, 1999.
- [3] Marcos Kawazoe Aguilera and Sam Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM J. Comput.*, 28(3):890–903, 1998.
- [4] Roberto Baldoni and Fabio Zito. Designing a service of failure detection in asynchronous distributed systems. In *Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 113–120, Magdeburg, Germany, May, 2001.
- [5] Marin Bertier, Olivier Marin, and Pierre Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proceedings of the International Conference on Dependable Systems and Networks*, Washington, D.C., USA, June 2002. IEEE Computer Society Press.
- [6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
- [7] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, New York, 2000. IEEE Computer Society Press.
- [8] M. Gouda and T. M. McGuire. Accelerated heartbeat protocols. In *Proceedings of the 18th International Conference on Distributed Computing Systems*. IEEE Computer Society Press, 1998.
- [9] I. Gupta, T. Chandra, and G. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proceedings of the 20th Annual ACM Symp. on Principles of Distributed Computing*, pages 170–179. ACM Press, 2001.
- [10] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. Technical report, Microsoft Research, Cambridge, UK, 2001.
- [11] Seungjae Han and Kang G. Shin. Experimental evaluation of behavior-based failure-detection schemes in real-time communication networks. *Transactions on Parallel and Distributed Systems*, pages 613–626, 1999.

- [12] Takuya Katayama Naohiro Hayashibara, Adel Cherif. Failure detectors for large-scale distributed systems. In *Proceedings of the International Workshop on Self-Repairing and Self-Configurable Distributed Systems*, Osaka, Japan, Oct. 2002.
- [13] Robbert Van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. In *Middleware '98: IFIP International Conference on Distributed Systems and Platforms and Open Distributed Processing*, pages 55–70. Springer Verlag, 1998.
- [14] A. Schiper. Failure detection vs. group membership in fault-tolerant distributed systems: Hidden trade-offs. In *Proceedings PAPM-PROBMIV 2002*, LNCS 2399, pages 1–15, Copenhagen, Denmark, July 2002. Springer Verlag. Invited talk.