Distributed Diagnosis in Dynamic Fault Environments

Arun Subbiah, *Student Member*, *IEEE*, and Douglas M. Blough, *Senior Member*, *IEEE*

Abstract—The problem of distributed diagnosis in the presence of dynamic failures and repairs is considered. To address this problem, the notion of bounded correctness is defined. Bounded correctness is made up of three properties: bounded diagnostic latency, which ensures that information about state changes of nodes in the system reaches working nodes with a bounded delay, bounded start-up time, which guarantees that working nodes determine valid states for every other node in the system within bounded time after their recovery, and accuracy, which ensures that no spurious events are recorded by working nodes. It is shown that, in order to achieve bounded correctness, the rate at which nodes fail and are repaired must be limited. This requirement is quantified by defining a minimum state holding time in the system. Algorithm HeartbeatComplete is presented and it is proven that this algorithm achieves bounded correctness in fully-connected systems while simultaneously minimizing diagnostic latency, start-up time, and state holding time. A diagnosis algorithm for arbitrary topologies, known as Algorithm ForwardHeartbeat, is also presented. ForwardHeartbeat is shown to produce significantly shorter latency and state holding time than prior algorithms, which focused primarily on minimizing the number of tests at the expense of latency.

Index Terms—Distributed diagnosis, dynamic failures, fault tolerance, synchronous systems.

1 Introduction

An important problem in distributed systems that are subject to component failures is the distributed diagnosis problem. In distributed diagnosis, each working node must maintain correct information about the status (working or failed) of each component in the system. In this paper, we consider the problem of achieving diagnosis despite dynamic failures and repairs. Previous work has almost exclusively dealt with the static fault situation wherein statuses of nodes remain fixed for as long as it takes an algorithm to completely diagnose the system. While a few works have attempted to consider dynamic events, no formal models have been developed and so correctness proofs and algorithm evaluations inevitably have reverted to the use of static models.

We present a formal model of dynamic behavior, which allows us to rigorously define what it means for a diagnosis algorithm to be correct in dynamic situations. This notion of correctness, referred to as bounded correctness, consists of three properties: bounded diagnostic latency, bounded start-up, and accuracy. For bounded diagnostic latency, all working nodes must learn about each event (node failure or repair) within a bounded time L. For bounded start-up, nodes that recover must determine a valid state for every other node within time S of entering the working state. Finally, accuracy ensures that no spurious events are recorded by any working node.

The specification of bounded correctness represents a strengthening of the properties required for a solution to the diagnosis problem. To our knowledge, no prior distributed diagnosis algorithm has been formally proven to achieve properties as strong as those of bounded correctness in the presence of truly dynamic failures and repairs. We present herein the first algorithms for distributed diagnosis that are rigorously proven to be correct in dynamic fault situations. Furthermore, evaluation of prior algorithms shows that our algorithms achieve significantly shorter diagnostic latency while tolerating substantially higher event rates than previous ones. We show in the next section that previous algorithms, by focusing almost exclusively on minimizing the number of tests performed, have unintentionally made themselves quite vulnerable to dynamic environments.

2 Related Work

The bulk of the work in system diagnosis has assumed a static fault situation [3], [9], [17], [18], [20], [23], i.e., the statuses of nodes do not change during execution of the diagnosis procedure. Some diagnosis algorithms, e.g., [5], [13], [19] allow dynamic failures and repairs to occur, but are only guaranteed to be correct when system status has become stable. One of the diagnosis algorithms in [4] assumes that nodes can fail dynamically, but cannot be repaired during execution of the diagnosis procedure. This approach is suitable in some systems, but is not a satisfactory solution in general. The diagnosis model of [14] considers dynamic failures but requires a centralized diagnosis entity.

Previous work on distributed diagnosis has focused almost exclusively on minimizing the number of tests performed. One interesting result of our work is to show that the goal of minimizing tests and the goal of effectively handling dynamic failures and repairs are directly in conflict. Prior algorithms that minimize the number of tests construct sparse testing graphs and propagate information in reverse direction of tests. The ideal testing property for which these algorithms strive is to have each node tested by exactly one

• *The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250. E-mail: {arun, dblough}@ece.gatech.edu.*

Manuscript received 22 Apr. 2002; revised 12 Mar. 2003; accepted 22 Sept. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 116387.

other node at each testing round. With dynamic failures and repairs, the latency and the minimum time a node must remain in a particular state can be as high as $(n-1)T$, where n is the number of nodes in the network and T is the duration of a testing round. Experiments in [5] on networked systems were conducted with parameters of $n = 60$ and $T = 30$ seconds, which yield a diagnostic latency and state holding time of about 30 minutes when dynamic environments are considered. The above result holds for all of the following algorithms: ADSD [5], BH [2], Adapt [24], and RDZ [19]. Other relevant algorithms such as Hi-ADSD and its variants [8], [9] have latencies of at least $\log_2^2 n$ rounds, which are still far greater than Algorithm HeartbeatComplete.

Since we assume testing is accomplished via heartbeat-based mechanisms which have low cost, we are not so concerned with the number of tests. Rather, we try to minimize diagnostic latency and state holding time in dynamic fault environments for both completely-connected and not-completely-connected networks. Our algorithm for completely-connected networks, known as Algorithm HeartbeatComplete, has a latency of approximately one heartbeat transmission round[1] and a state holding time of about half a round. Our algorithm for not-completely-connected networks, known as Algorithm ForwardHeartbeat, has both latency and state holding time equal to approximately one round. Thus, both algorithms are capable of very fast propagation of events and they work in highly dynamic environments.

A problem closely related to distributed diagnosis, known as the synchronous group membership problem [6], [7], [11], [12], [15], is for each working node to maintain correct information about the group of working nodes with which it can communicate, and for all nodes in one group to agree on the membership of the group. In [12], it is shown that, under some models, these two problems are equivalent and an algorithm for one problem can be converted to an algorithm for the other. The primary difference between the two approaches is the requirement of agreement among nodes in the group membership problem, while agreement is not required in distributed diagnosis.

Our work is closest to the approach taken in [6], where there is no limit on how many nodes can change state during execution of the algorithm, but there is a limit on how frequently an individual node can change state. The algorithm of [6] guarantees that working nodes make identical membership changes at the same local clock times. To achieve this strong property, the algorithm requires synchronized clocks and a form of atomic broadcast. Bounded correctness can be achieved without clock synchronization and does not require any special communication mechanisms. In addition, the work of [6] does not derive lower bounds on state holding time and latency and, therefore, does not address the limits of dynamic behavior nor the optimality of the presented algorithm.

Another related area is that of failure detection. Failure detectors are used to solve higher-level problems such as consensus and atomic broadcast in asynchronous and partially-synchronous systems. To our knowledge, the latest work that considers failures and recoveries is [1]. In [1], existence of nodes that eventually are permanently working or permanently failed is assumed. Since we do not assume existence of such nodes in our model, all nodes are unstable in the terminology of [1]. In [1], working nodes do not distinguish between unstable and failed nodes. Hence, no evaluation is done of the minimum time an unstable node needs to be in a particular state so that its status is accurately tracked. Diagnosis done by unstable nodes on other unstable or permanently failed nodes is also not considered. Last, [1] considers only asynchronous completely-connected networks.

1. This is equivalent to a testing round in our approach.

3 System Model

In this section, we present some basic definitions used throughout the paper.

3.1 Communication Model

Diagnosis algorithms can use either unicast or multicast communication. We assume generic parameters that could apply to either type of communication. We also assume a *synchronous* system in which the communication delay is bounded. This is an implicit assumption in all prior work on distributed diagnosis.

Definition 1. *The send initiation time, $\Delta_{\text{send\_init}}$, is the time between a node initiating a communication and the last bit of the message being injected into the network. This includes message set-up time on the node, any delay in accessing the communication medium, and the time to inject all of the message bits into the network. To simplify analysis, it is assumed that $\Delta_{\text{send\_init}}$ is a constant.*

Definition 2. *The minimum and maximum message delays, $\Delta_{\text{send\_min}}$ and $\Delta_{\text{send\_max}}$, are the minimum and maximum times, respectively, between the last bit of a message being injected into the network and the message being completely delivered at a working neighboring node.*

We assume that messages are encoded in such a way, e.g., using checksums, to enable incomplete messages to be detected and discarded. Hence, failures that occur on a sending node in the middle of a message transmission (prior to the last bit of the message, including the checksum, being injected into the network) appear as omissions at receiving nodes.

We consider both completely-connected and not-completely connected networks. In a completely-connected network, there is a direct communication channel between every pair of nodes. It is not difficult to show that this is a requirement to be able to achieve bounded correctness with an arbitrary number of node failures. In not-completely connected networks, intermediate nodes relay messages between some source-destination pairs. Hence, the number of node failures is limited such that the network remains connected at all times.

3.2 Fault Model

We consider crash[2] faults in nodes. The network delivers messages reliably. The crash fault assumption differs from traditional work on system-level diagnosis for which the fault model is not specified. However, the classical assumption that tests are perfect implies some class of

2. Fail silent [16], manifest [21], fail-stop [22], and benign [26] faults can also be modeled as crash faults.

easily-detectable faults. The crash fault assumption is necessitated by our use of heartbeat-based algorithms for diagnosis, which have been more commonly used in group membership algorithms. Hiltunen [12] shows how heartbeat-based algorithms can be transformed into test-based algorithms and vice versa. Using this transformation, our algorithms could be easily converted to ones that use explicit testing and the crash fault assumption could then be loosened.

Nodes can alternate between working correctly and being crashed in our model.[3] Hence, the status of a node is modeled by a state machine with two states, *failed* and *working*. Failed nodes do not send messages nor do they perform any computation. Working nodes execute faithfully the diagnosis procedure.

Definition 3. *The* state holding time *is the minimum time that a node remains in one state before transitioning to the other state.*

It is important to note that, in completely-connected networks, Definition 3 is our model's only restriction on the timing of node failures and repairs. Since node failures and repairs are independent in this model, there are no restrictions on the number of nodes that are in the failed state at any one time nor on the number that can fail (or recover) at the same instant. This model is, therefore, considerably less restrictive than many of the models used in prior work. Since we will show later that a nonzero state holding time is required to solve the problems of interest and this is the only assumption on failure timing in our model, it is the least restrictive dynamic model possible for this problem.

For not-completely-connected networks, we define the connectivity of the network as follows.

Definition 4. *The connectivity of the network, k, is the minimum number of nodes, the removal of which can cause the network to become disconnected.*

For these networks, we assume that fewer than k nodes are in the failed state at any given instant of time.

3.3 Time and Clock Models

Since we are interested in dynamic failure situations in which failure and recovery timing is critical, it is imperative that the notion of time be well defined.

Definition 5. *Time that is measured in an assumed Newtonian time frame (which is not directly observable) is referred to as* real time.

Definition 6. *Time that is directly observable on a node's clock is referred to as* clock time. *The clock time of node X at real time t is denoted by $T_X(t)$.*

Definition 7. *While a node is in the working state, its clock experiences bounded drift. This means that if a node X is in the working state continuously during a real-time interval $[t_1, t_2]$, then for all real-time intervals $[u_1, u_2] \subseteq [t_1, t_2]$*

$$|[T_x(u_2) - T_x(u_1)] - (u_2 - u_1)| \leq \rho(u_2 - u_1),$$

where $\rho << 1$ is the maximum drift rate of a clock.

3. We use the term *dynamic fault model*, which is standard in the diagnosis literature. The model is referred to as the *crash-recovery* model when used in the study of other distributed algorithms.

3.4 Algorithm Assumptions

We assume algorithms work by use of heartbeat messages, i.e., each node periodically initiates a round of message transmissions to other nodes in order to indicate that it is working.

Definition 8. *Assume an arbitrary node X initiates a round of heartbeat transmissions at real time t and remains in the working state indefinitely afterward. X will initiate another round of heartbeat transmissions no later than real time $t + (1 + \rho)\pi$, where π is the* heartbeat period.

We do not restrict algorithms to exchange status information *only* by heartbeat messages. For example, a node could send heartbeat messages to a subset of other nodes and then rely on those nodes to relay the information that it is working to the remaining nodes via ordinary (nonheartbeat) messages. We do assume, however, that heartbeats are the basic mechanism for a node to notify other nodes that it is working.

After a node recovers, it could initiate a round of heartbeats immediately after entering the working state or it could wait before doing so. If the node waits, however, it should not wait more than π in local clock time in order to maintain the heartbeat period. This leads to the following definition.

Definition 9. *The recovery wait time for an algorithm, denoted by $W \leq \pi$, is the local clock time for which the algorithm waits after entering the working state before initiating a round of heartbeat transmissions.*

3.5 Bounded Correctness

In a system that dynamically experiences failures and repairs and has nonzero communication delay, the view that any node has of the system at any time is, inevitably, out of date. To examine the limits of diagnosis algorithms, we consider what we believe are the weakest properties that any such algorithm should guarantee. Each working node should have timely information about the status of every other node, either working or failed, in the system. Any transition between the two states on a node is referred to as an *event*. The goal is for working nodes to learn about every event in the system as quickly as possible, to have their views of other nodes be out of date by only a bounded amount, and to not detect any spurious events.

Formally, we represent this goal by three properties which we collectively refer to as *bounded correctness*. Specification of one of these properties requires the following definition.

Definition 10. *A state held by a working node X for another node Y at time t is said to be T-valid if node Y was in the indicated state at some point during the interval $[t - T, t]$.*

Property 1: Bounded Diagnostic Latency. *Consider any event in the system that occurs at an arbitrary real time t. Any node that is in the working state continuously during the interval $[t, t + L]$ learns about the event and records the new state of the node that experienced the event by time $t + L$, where L is an algorithm-dependent bounded time referred to as the* diagnostic latency *of the algorithm.*

Property 2: Bounded Start-Up. *Consider the recovery of an arbitrary node X at real time t. If X remains in the working *state continuously during the interval $[t, t+S]$, then at time $t+S$, X holds L-valid states for every other node in the system, where $S \geq L$ is an algorithm-dependent bounded time referred to as the* start-up time *of the algorithm.*

Property 3: Accuracy. *Consider an arbitrary working node X after its start-up time. Every state transition (working to failed or failed to working) recorded by X for an arbitrary node Y corresponds to an actual event that occurred on Y and no single event on Y causes multiple state transitions to be recorded on X.*

Taken together, these properties ensure that after a node recovers (or starts up for the first time), it will determine valid state information about every other node in the system within bounded time and from that time on it will maintain a faithful record of events that occur on all nodes. Bounded Diagnostic Latency ensures that no events are missed, while Accuracy guarantees that no spurious events are recorded.

4 Completely Connected Networks

4.1 Limits on Algorithm Performance

In this section, we derive lower bounds on the diagnostic latency, start-up time, and state holding time achievable by any heartbeat-based diagnosis algorithm in completely-connected networks. The maximum time between two consecutive heartbeats arriving from a continuously working node at any other node in the system sets a limit on how early failed nodes can be identified by the absence of a heartbeat. This is specified in the following lemma.

Lemma 1. *Assume an arbitrary node Y is in the working state continuously for sufficiently long to send two consecutive heartbeats to another arbitrary node X. The maximum time between the two heartbeats arriving at X, $\Delta_{\mathrm{heartbeat}}$, is $(1+\rho)\pi + \Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}$.*

Proof. Suppose Y initiates the first heartbeat at time t. That heartbeat will arrive at X at time $t + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_min}}$ at the earliest. By Definition 8, Y will initiate its next heartbeat at time $t + (1+\rho)\pi$ at the latest. This heartbeat will arrive at X at time $t + (1+\rho)\pi + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_max}}$ at the latest. Subtracting the earliest and latest arrival times on X yields the maximum heartbeat interarrival time stated in the lemma. □

Lemmas 2 and 3 provide the desired performance limits for any algorithm.

Lemma 2. *The diagnostic latency and start-up time of any heartbeat algorithm that achieves bounded correctness are both at least $(1+3\rho)\pi + 2(1+\rho)\Delta_{\mathrm{send\_max}} - (1+2\rho)\Delta_{\mathrm{send\_min}}$.*

Proof. The worst-case latency for the detection of node Y's failure by node X occurs if Y fails immediately after initiating a heartbeat to X. If t represents the heartbeat initiation time, Y will fail at time $t + \Delta_{\mathrm{send\_init}}$ and the heartbeat will be received by X at time $t + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_max}}$ at the latest. Factoring in clock drift, X must then wait $(1+\rho)\Delta_{\mathrm{heartbeat}}$ time on its local clock without receiving a heartbeat before concluding that Y has failed. Subtracting the failure time from the latest detection time and simplifying yields a maximum failure detection latency of $(1+3\rho)\pi + 2(1+\rho)\Delta_{\mathrm{send\_max}} - (1+2\rho)\Delta_{\mathrm{send\_min}}$.

We now analyze the latency in detecting a node's recovery. We assume that nodes initiate heartbeat transmission immediately after making a transition from the failed state to the working state because this produces the shortest possible latency for recovery detection. The maximum delay before a working node receives a heartbeat from a recovered node is, therefore, $\Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_max}}$. Since $\Delta_{\mathrm{send\_init}} < \pi$, this is shorter than the failure detection latency derived above. Hence, the latency is equal to the failure detection latency.

For start-up time, we also need to consider the amount of time it takes after a node X returns to the working state before it determines an initial status for each other node in the system. The question here is, how long must node X wait without receiving a heartbeat from node Y before concluding that node Y is failed? The worst-case occurs if a heartbeat arrived from node Y just prior to node X's recovery. In this case, Y must wait for clock time $(1+\rho)\Delta_{\mathrm{heartbeat}}$ before it is safe to conclude that node Y failed. In real time, this could take as long as $(1+\rho)(1+\rho)\Delta_{\mathrm{heartbeat}}$. Since this is less than the minimum diagnostic latency and start-up time cannot be less than latency, the minimum start-up time is equal to the minimum latency derived above. □

Lemma 3. *The state holding time for any heartbeat algorithm to achieve bounded correctness is at least $(1+4\rho)\pi/2 + (1+2\rho)(\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) - \rho\Delta_{\mathrm{send\_init}}$.*

Proof. We analyze the minimum state holding time as a function of the recovery wait time W and then minimize the function with respect to W to determine the minimum for any algorithm. As in the proof of Lemma 2, we ignore ρ^2 terms. Denote the minimum possible state holding time by $\mathrm{SHT}_{\min}$.

After a node Y recovers from the failed state, it must send a single heartbeat so that other nodes detect the recovery event. Hence,

$$\mathrm{SHT}_{\min} > (1+\rho)W + \Delta_{\mathrm{send\_init}}. \tag{1}$$

This is the minimum time a node must remain in the working state after making a transition into that state.

Now, consider the minimum time a node must remain in the failed state in order for the transition into that state to be detected. Such a transition cannot be detected if the node returns to the working state and sends a new round of heartbeats as early as if it had never left the working state. The worst-case is if a node fails at time $t + \Delta_{\mathrm{send\_init}}$ immediately after successfully initiating a heartbeat to another node X and returns to the working state in time $\mathrm{SHT}_{\min}$. In this situation, the first heartbeat arrives at X at time $t + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_max}}$ at the latest. The second heartbeat will arrive at X at time $t + \Delta_{\mathrm{send\_init}} + \mathrm{SHT}_{\min} + (1-\rho)W + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_min}}$ at the earliest. If the difference between these arrival times is no greater than $(1+2\rho)\Delta_{\mathrm{heartbeat}}$, then node X cannot distinguish this situation from one in which node Y remained in the working state for the entire interval. This yields

$$\begin{aligned}\mathrm{SHT}_{\min} > {} & (1+3\rho)\pi + 2(1+\rho)(\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) \\ & - \Delta_{\mathrm{send\_init}} - (1-\rho)W.\end{aligned} \tag{2}$$

From (1) and (2), we have

$$\mathrm{SHT}_{\min} > \max((1+\rho)W + \Delta_{\mathrm{send\_init}}, (1+3\rho)\pi + 2(1+\rho)\times (\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) - \Delta_{\mathrm{send\_init}} - (1-\rho)W). \quad (3)$$

As W increases, the right-hand side of (1) monotonically increases and the right-hand side of (2) monotonically decreases. Furthermore, since $\Delta_{\mathrm{send\_init}} \ll \pi$ in practice, the right-hand side of (1) is less than the right-hand side of (2) when $W = 0$. This means that the two expressions cross for some $W > 0$. Thus, the right-hand side of (3) is minimized when the right-hand sides of (1) and (2) are equal. Setting these two expressions to be equal and solving for W yields

$$W = (1+3\rho)\pi/2 + (1+\rho)(\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) - \Delta_{\mathrm{send\_init}}, \quad (4)$$

and the lemma follows.

Note that it is theoretically possible for the right-hand side of (4) to be greater than π, which is outside the allowable range for W. However, in practice, all the parameters in (4) are small compared to π and the value for W that produces the smallest possible state holding time is approximately $\pi/2$. Even if the right-hand side of (4) is greater than π, then from (3), $\mathrm{SHT}_{\min}$ in the range $0 \le W \le \pi$ will be greater than its absolute minimum and the lemma will still hold. □

If we assume that ρ, $\Delta_{\mathrm{send\_max}}$, $\Delta_{\mathrm{send\_min}}$, and $\Delta_{\mathrm{send\_init}}$ are all much smaller than π, which is likely to be true in practice, then Lemma 3 gives the minimum state holding time as approximately $\pi/2$. This is somewhat counter-intuitive in that it would seem that a node should remain in the failed state for at least π time before recovering in order to guarantee that its failure is observed by all working nodes. However, this analysis allows for the possibility that an algorithm forces nodes that recover to delay sending their heartbeats in order to extend the interarrival times of their heartbeats on working nodes by enough to allow those nodes to observe the failure.

4.2 An Optimal Algorithm for Bounded Correctness in Completely-Connected Networks

In this section, we present a new heartbeat-based algorithm, referred to as Algorithm HeartbeatComplete, for distributed diagnosis that provably minimizes diagnostic latency, start-up time, and state holding time in completely-connected networks.

4.2.1 Description of Algorithm HeartbeatComplete

The analysis of Section 4.1 shows that, if heartbeat messages are sent periodically by each working node to all other nodes with a period of π, then it is safe to declare a node to be failed when no heartbeat is received from it within a clock time of $(1+\rho)\Delta_{\mathrm{heartbeat}}$. The pseudocode for Algorithm HeartbeatComplete, which makes use of this fact is shown in Fig. 1.

Each node executing Algorithm HeartbeatComplete uses a broadcast mechanism to send heartbeat messages to all of its neighbors in a single message requiring only one send initiation. Due to the assumption that faults are restricted to nodes, if a node successfully initiates a heartbeat, it will be received by all of its neighbors within time $\Delta_{\mathrm{send\_max}}$. However, it should be emphasized that there are no ordering requirements in this broadcast, e.g., it is neither causal nor atomic, so that broadcasts sent by two different nodes can be received in different orders on different nodes.

```
Upon entering the working state, node $X$ executes:

Status[$X$] ← working;
SetSendHeartbeatTimer($W$);
for $Y = 0$ to $n - 1$ do
    if $Y \neq X$ then
        Status[$Y$] ← unknown;
        SetReceiveHeartbeatTimer$_Y$($(1+\rho)\Delta_{\mathbf{heartbeat}}$);
    endif;
endfor;

Upon receiving a heartbeat message from node $Y$, node $X$ executes:

Status[$Y$] ← working;
SetReceiveHeartbeatTimer$_Y$($(1+\rho)\Delta_{\mathbf{heartbeat}}$);

Handlers execute when their corresponding timers expire:

Procedure SendHeartbeatTimerHandler
    broadcast heartbeat message to all neighbors;
    SetSendHeartbeatTimer($\pi$);

Procedure ReceiveHeartbeatTimerHandler$_Y$
    Status[$Y$] ← failed;
```

Fig. 1. Pseudocode for Algorithm HeartbeatComplete.

4.2.2 Algorithm Analysis

The following theorem states that Algorithm Heartbeat-Complete achieves bounded correctness and characterizes its diagnostic latency, start-up time, and state holding time. The proof can be found in the appendix.

Theorem 1. *With $W \le \pi$ and a state holding time of*

$$\max((1+\rho)W + \Delta_{\mathrm{send\_init}}, (1+3\rho)\pi + 2(1+\rho) (\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) - \Delta_{\mathrm{send\_init}} - (1-\rho)W),$$

Algorithm HeartbeatComplete achieves bounded correctness with diagnostic latency and start-up time equal to

$$\max((1+3\rho)\pi + 2(1+\rho)\Delta_{\mathrm{send\_max}} - (1+2\rho)\Delta_{\mathrm{send\_min}}, (1+\rho)W + \Delta_{\mathrm{send\_init}} + \Delta_{\mathrm{send\_max}}).$$

Corollary 1. *With*

$$W = \min(\pi, (1+3\rho)\pi/2 + (1+\rho) (\Delta_{\mathrm{send\_max}} - \Delta_{\mathrm{send\_min}}) - \Delta_{\mathrm{send\_init}}),$$

Algorithm HeartbeatComplete achieves bounded correctness with minimum diagnostic latency and start-up time while requiring minimum state holding time.

Proof. With W as specified in the corollary, the value of the diagnostic latency and start-up time given by Theorem 1 becomes $(1+3\rho)\pi + 2(1+\rho)\Delta_{\mathrm{send\_max}} - (1+2\rho)\Delta_{\mathrm{send\_min}}$, which, according to Lemma 2, is the minimum possible.

The expression for state holding time given in Theorem 1 is exactly the one derived in the proof of Lemma 3. This expression was shown to be minimized by the value of W specified in the corollary. □

Note that Algorithm HeartbeatComplete does not specify any value for W. The above corollary gives the value of W that minimizes the diagnostic latency and start-up time while requiring minimum state holding time. If minimizing the minimum time a node must spend in the working state is critical, then W should be set to zero.

4.2.3 Message Cost of Algorithm HeartbeatComplete

Many systems that are logically completely-connected actually employ a bus or redundant bus structure. In such systems, HeartbeatComplete sends one heartbeat message per node per round, which is the minimum possible message cost. Even if buses are not used, a single broadcast message per node per round is generated. In networks with efficient broadcast support, e.g., Ethernet, this cost can still be low. In a complete network made entirely of point-to-point links, the algorithm would generate $n(n-1)$ messages per round, which is higher than the best known algorithms for completely-connected networks; $2n$ ([5]) and $2n \log n$ ([8] and [9]). However, the cost of HeartbeatComplete still represents only one message per link per node per round in this case. Note that, to satisfy our model, any type of logical completely-connected network must reliably deliver all messages, and so redundant buses, redundant links, or reliable broadcast support must be provided.

5 Not-Completely-Connected Networks

5.1 Description of Algorithm ForwardHeartbeat

In this paper, we consider a restricted case where the maximum number of nodes that can be in the failed state at any time instant is $(k-1)$, where k is the connectivity of the network as defined in Section 3, thereby ensuring that the network remains connected at all times.

Algorithm ForwardHeartbeat follows the general principles of Algorithm HeartbeatComplete where working nodes perform their own independent diagnoses of the system. Working nodes periodically send heartbeats, which are propagated throughout the network. Here, a heartbeat, which should be understood at the conceptual level, consists of several heartbeat messages that may be propagating through different parts of the network at a given instant of time. A node initiates a heartbeat by sending heartbeat messages on all its links. These heartbeat messages, when received by working neighboring nodes, are in turn sent on all their links and so on resulting in the heartbeat being propagated throughout the network.

When node X receives a new heartbeat from node Y, node X stores the heartbeat in a buffer replacing any older heartbeats from node Y. If node X times out waiting for node Y's next heartbeat, then node Y's heartbeat is removed from node X's buffer and node Y is diagnosed as faulty by node X. Hence, the presence of node Y's heartbeat in node X's buffer indicates node X believes that node Y is working. If a neighbor of node X recovers, then node X sends the newly-recovered neighboring node all heartbeats stored in its buffer. This ensures propagation of heartbeats in a dynamic fault environment.

A heartbeat message consists of the following fields:

1. Node_id: The ID of the node that initiated the heartbeat.
2. Seq_no: The physical sequence number of the heartbeat.
3. delay: The minimum time the heartbeat message was in the network before being received.

The Seq_no field serves to distinguish successive heartbeats from the same node. The first heartbeat sent by a node after its recovery has sequence number 0, the next heartbeat has sequence number 1, and so on. When the maximum sequence number (MAX_SEQ_NUM) that can be stored in this field is exceeded, the sequence number is wrapped around to 0. However, the logical sequence number is a monotonically increasing quantity. Unless otherwise mentioned, throughout this paper, sequence numbers denote logical sequence numbers. This simplification will hold as long as MAX_SEQ_NUM satisfies a certain lower bound, proven in Lemma 12. The Node_id and Seq_no fields together identify the heartbeat.

When a node Y initiates a new heartbeat, it initializes the delay field to the minimum delay that will be encountered before the heartbeat messages could reach its neighboring nodes and then sends them out. Also, at the same time, node Y stores this new heartbeat in its local buffer with the delay field set to zero. Nodes keep track of the amount of time each heartbeat is stored in their buffers. For a heartbeat stored locally in the originating node, the delay field will always be 0. When a node retransmits or relays a heartbeat, it adds to the delay field the length of time the heartbeat was stored in its buffer and the minimum time it takes to traverse the next hop to reach the neighboring node before sending it out. Thus, a node keeps track of the minimum length of time the heartbeats stored in its buffer have existed in the network. The significance of this will be evident in Lemma 7.

Each node maintains an array of n entries, where n is the total number of nodes in the network. The ith entry in the array stored in node X contains:

1. Status: Node X's view of node i. Possible values are $failed$, $working$, and $unknown$.
2. Last_seq_no: The Seq_no field of the newest heartbeat received from node i by node X.

Algorithm HeartbeatComplete is used to diagnose neighboring nodes by having a node X discard heartbeats belonging to a neighboring node Y that do not arrive on the link connecting X and Y. From now, we focus only on the diagnosis of nonneighboring nodes.

Due to the existence of multiple paths between nodes, it is possible that a node times out thereby detecting a fault event and then receives a stale heartbeat. Arrival of this stale heartbeat does not indicate a recovery event. Hence, when nodes time out, they discard any heartbeats they may receive from the node just diagnosed to be faulty for a predetermined amount of time called the heartbeat rejection time, defined below. As a result, for a genuine recovery event to be detected, the failed state holding time should be made sufficiently large to guarantee that no new heartbeat message arrives during the rejection time.

Definition 11. *The Heartbeat Rejection Time, denoted by T_{reject}, is the period of time a node X discards heartbeats from a node Y after diagnosing node Y to be faulty.*

```
Upon entering the working state, node $X$ executes:

Node[$X$].Status ← working;
Node[$X$].Last_seq_no ← −1;
Clear Buffer;
for $Y = 0$ to $n-1$, $Y \neq X$ do
    Node[$Y$].Status ← unknown;
    Node[$Y$].Last_seq_no ← −1;
endfor;
SetSendHeartbeatTimer($W$);
SetStartupTimer($(1+\rho)(W+T_{\text{exist}})$);

Upon receiving a heartbeat message, node $X$ executes:

$Y$ ← ReceivedHeartbeat.Node_id;
if ((ReceivedHeartbeat.Seq_no > Node[$Y$].Last_seq_no) and (not)($\exists\ T_{\text{reject}}\text{Timer}_Y$)) then
    if (node $Y$ is not a neighbor of node $X$) then
        SetReceiveHeartbeatTimer$_Y$($T_{\text{timeout}}$);
        Node[$Y$].Status ← working;
        Node[$Y$].Last_seq_no ← ReceivedHeartbeat.Seq_no;
        Buffer[$Y$] ← ReceivedHeartbeat;
        Add ($\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$) to ReceivedHeartbeat.delay;
        Send ReceivedHeartbeat on all links except the link on which it arrived;
    else
        if (heartbeat received on link connecting node $Y$) then
            SetReceiveHeartbeatTimer$_Y$($(1+\rho)((1+\rho)\pi + \Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$);
            Node[$Y$].Last_seq_no ← ReceivedHeartbeat.Seq_no;
            Buffer[$Y$] ← ReceivedHeartbeat;
            if (Node[$Y$].Status ≠ working) then
                Add $(1-\rho)$(amount of time heartbeat was kept buffered) and ($\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$)
                to the delay field in all heartbeats stored in buffer and send them to node $Y$;
            endif;
            Node[$Y$].Status ← working;
            Add ($\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$) to ReceivedHeartbeat.delay;
            Send ReceivedHeartbeat on all links except the link on which it arrived;
        endif;
    endif;
endif;

Handlers execute when the corresponding timers expire:

Procedure SendHeartbeatTimerHandler$_X$
    New_Heartbeat.Node_id ← $X$;
    New_Heartbeat.Seq_no ← Node[$X$].Last_seq_no + 1;
    if (New_Heartbeat.Seq_no > MAX_SEQ_NUM) then
        New_Heartbeat.Seq_no ← 0;
    New_Heartbeat.delay ← 0;
    Buffer[$X$] ← New_Heartbeat;
    Add ($\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}$) to New_Heartbeat.delay and send New_Heartbeat on all links;
    SetSendHeartbeatTimer($\pi$);

Procedure ReceiveHeartbeatTimerHandler$_Y$
    Set $T_{\text{reject}}\text{Timer}_Y(T_{\text{reject}})$;
    Node[$Y$].Status ← failed;
    Node[$Y$[.Last_seq_no ← −1;
    Buffer[$Y$] ← null;

Procedure StartupTimerHandler$_X$
    for $Y = 0$ to $n-1$, $Y \neq X$, do
        if (Node[$Y$].Status = unknown) then
            Node[$Y$].Status ← failed;
            Node[$Y$[.Last_seq_no ← −1;
            Buffer[$Y$] ← null;
        endif;
    endfor;
```

Fig. 2. Pseudocode for Algorithm ForwardHeartbeat.

The pseudocode for Algorithm ForwardHeartbeat is shown in Fig. 2. T_{exist} and T_{timeout} are defined in the next section.

5.2 Analysis of Algorithm ForwardHeartbeat

We find it useful to distinguish between the working and the failed state holding times in case of Algorithm ForwardHeartbeat. Hence, we have the following definitions.

Definition 12. *The* working state holding time, *denoted by SHT_w, is the minimum time a node remains in the working state before transitioning to the failed state.*

Definition 13. *The* failed state holding time, *denoted by SHT_f, is the minimum time a node remains in the failed state before transitioning to the working state.*

In addition, we have the following definition.

Definition 14. *The maximum number of neighbors of any node is denoted by d.*

A brief overview of the analysis is as follows: We first prove that W must be zero in order to minimize the latency and state holding times. We then derive message transmission bounds: the minimum and maximum time it takes for a heartbeat to reach all nodes that are continuously working. Based on these values, the length of time a node waits before timing out and diagnosing a node to be faulty ($= (1+\rho)\Delta_{\text{heartbeat}}$ in the completely-connected case) is computed. We next calculate the maximum time a heartbeat message can exist in the network, a parameter necessary for further analysis. We then derive the state holding times and T_{reject} defined earlier. Finally, we prove that with the derived values of state holding times, Algorithm Forward-Heartbeat satisfies the bounded correctness properties.

Lemma 4. *The minimum diagnostic latency, staleness, and the state holding times for Algorithm ForwardHeartbeat are achieved when* $W = 0$.

Proof Sketch. An intuitive reasoning is given here due to space restrictions. For a complete proof, see [25].

Consider the way heartbeat messages are propagated in the presence of node failures and recoveries. Heartbeat messages must be guaranteed to reach all nodes that were in the working state since heartbeat initiation. Upon detecting the recovery of a neighboring node, nodes send heartbeats stored in their buffers to that neighboring node. The freshly recovered node passes them on and messages get propagated. With $W > 0$, this will clearly lead to a high value for heartbeat message propagation to all nodes in the system. Hence, SHT_w is required to be high so that there is adequate time for a freshly recovered node to "pump in" its heartbeat to all nodes in the network. Since a node is required to be in the failed state until all working nodes timeout after receiving the last heartbeat sent, the failed state holding time will also increase with W. Hence, for the best case, W must be zero. □

Lemma 5. *The minimum time between a node initiating a heartbeat and a heartbeat with the same or a higher sequence number from the same node being received at some nonneighboring node that is working since heartbeat initiation, denoted by* $D_{\min}$*, is* $2(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$.

Proof. Propagation time for a heartbeat from node Y to nonneighboring node X will be minimal if X and Y are separated by only one node. $D_{\min}$ is then

$$2(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}}).$$

□

Lemma 6. *The maximum time between a node initiating a heartbeat and a heartbeat with the same or a higher sequence number from the same node being received by all nonneighboring nodes that are working since heartbeat initiation, denoted by* D_{maxn}*, is* $d(k-1)(n-1)\ \Delta_{\text{send\_init}} + (n+k-2)(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - (k-1)\epsilon$*, where* ϵ *is an arbitrary small positive constant, if the failed state holding time,* SHT_f*, is at least* D_{maxn}.

Proof. Since the failed state holding time, SHT_f, is assumed to be at least D_{maxn}, any node that fails after a node Y initiates a heartbeat cannot recover until all nodes that are in the working state since heartbeat initiation receive the heartbeat from node Y. Further, since the number of nodes that can be in the failed state is limited to $k-1$, the worst-case for calculating D_{maxn} is arrived at by having only k neighbors for node Y.

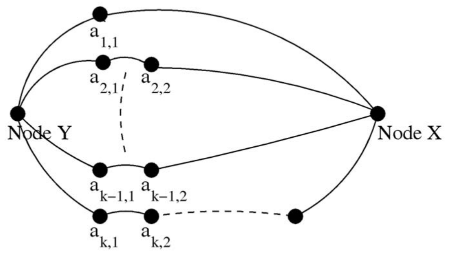


Fig. 3. Network for Proof of Lemma 6.

The network considered for constructing the worst-case is shown in Fig. 3. A network with connectivity k has at least k disjoint paths between any two nodes in the network. Here, X and Y have k disjoint paths between them, with node Y having only k neighboring nodes. At real time t, $(k-1)$ neighbors of node Y ($a_{2,1}, a_{3,1}, \ldots, a_{k,1}$) are in the failed state for time greater than SHT_f. Node Y initiates a heartbeat at time t, which is received only by node $a_{1,1}$. Node $a_{1,1}$ fails just before it could forward the complete heartbeat to its neighboring nodes, and at that same instant one of the failed neighboring nodes of Y (node $a_{2,1}$) recovers. This node sends out a heartbeat indicating its recovery which prompts node Y to forward its buffered heartbeats to it. $a_{2,1}$ cannot fail before it could forward these heartbeats and $a_{2,2}$ receives it. In the worst-case, $a_{2,1}$ could have the $(k-1)$ failed nodes as its neighbors, and one other working node ($a_{2,2}$) which is not a neighbor of node Y.

Let $a_{2,2}$ fail just before it could forward node Y's heartbeat to its neighbors, and at that same instant one of the failed nodes that is a neighbor of node Y ($a_{3,1}$) recovers. This process repeats until the kth neighbor of node Y ($a_{k,1}$) recovers. No more nodes can fail because all the $k-1$ failed nodes at this point cannot recover until D_{maxn} time has elapsed.

We assume that the heartbeat that propagates through the kth neighboring node of node Y passes through all the remaining working nodes in the system. This is possible if the remaining working nodes form a linear chain between them, with each working node having its remaining $k-1$ neighbors as the nodes that are currently in the failed state. The worst-case is thus arrived at by having as many failures as possible during the heartbeat propagation and by having the heartbeat reach all other nodes in the network before reaching the final destination node.

Hence, D_{maxn} reduces to the following expression:

$$\begin{aligned}&\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + \Delta_{\text{send\_init}} - \epsilon + (k-2)[d(n-1)\\ &\Delta_{\text{send\_init}} + 3(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}) - \epsilon] + d(n-1)\\ &\Delta_{\text{send\_init}} + 2\Delta_{\text{send\_init}} + 3\Delta_{\text{send\_max}} + (n-2k)\\ &(\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}).\end{aligned}$$

Simplifying the above expression for D_{maxn} gives the expression stated in the lemma. □

Lemma 7. *Upon receipt of a new heartbeat, the period of time a node that is continuously in the working state waits for the next new heartbeat before detecting a fault event, called the timeout period* T_{timeout}*, is* $(1+2\rho)\pi + (1+\rho)(D_{\text{maxn}} - Heartbeat.delay)$, *where Heartbeat.delay is the value stored in the delay field of the last heartbeat received.*

Proof. Let node Y initiate a heartbeat at real time t. That heartbeat will arrive at node X at real time $t + Heartbeat.delay$ at the earliest. Node Y initiates its next heartbeat at real time $t + (1+\rho)\pi$, which will reach node X at real time $t + (1+\rho)\pi + D_{\text{maxn}}$ at the latest if node X is in the working state at the time the heartbeat is initiated. Node X should not have timed out in this interval. Hence, the timeout period T_{timeout} is $(1+\rho)[(1+\rho)\pi + D_{\text{maxn}} - Heartbeat.delay]$. Since ρ is assumed to be very small, this reduces to

$$(1+2\rho)\pi + (1+\rho)(D_{\text{maxn}} - Heartbeat.delay).$$

□

Lemma 8. *The maximum possible time a heartbeat can exist in the network, denoted by* T_{exist}, *is* $(1+3\rho)\pi + (1+2\rho)D_{\text{maxn}} + n(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$. *A heartbeat is said to exist in the network if either the heartbeat is propagating in the network or the heartbeat is stored in some node's buffer.*

Proof. The worst-case is when node Y fails sometime after sending a heartbeat so that the heartbeat issued is removed by having nodes timeout waiting for the next heartbeat from node Y. Let the last heartbeat issued by node Y at time t be received by node Z which is in the working state since time t with a delay D. Hence, node Z will timeout waiting for the next heartbeat from node Y at time $t_1 = t + D + (1+3\rho)\pi + (1+2\rho)(D_{\text{maxn}} - Heartbeat.delay)$ at the latest. Consider another node X in the network such that one of the paths connecting nodes X and Z has $k-1$ nodes in it, with all these $k-1$ nodes in the failed state. Let the $k-1$ nodes be denoted by $a_1, a_2, \ldots, a_{k-1}$. At time $t_1 - k(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$, node Z receives a heartbeat from node a_1 indicating the recovery of node a_1. Node Z forwards the buffered heartbeat to node a_1, adding $(\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})$ to the $Heartbeat.delay$ field. Node a_1 forwards the heartbeat to node a_2 because it just received a heartbeat from node a_2 indicating node a_2's recovery. Assuming the same situation repeats itself for the remaining $k-2$ failed nodes, node X will receive a buffered heartbeat regarding node Y at time $t + T_{\text{exist}} = t + D + (1+3\rho)\pi + (1+2\rho)(D_{\text{maxn}} - Heartbeat.delay) + k(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$.

$D - Heartbeat.delay$ will be maximum under the following circumstances. Assume node Y has only k neighboring nodes. Consider the k disjoint paths connecting node Y and node Z. At real time t, all but one of the k neighboring nodes of node Y is in the working state. The path between node Y and node Z that has the working neighboring node of node Y has $n-k-1$ nodes in it. Node Y initiates a heartbeat at real time t, which is propagated right away through the $n-k-1$ nodes to node Z, with the communication delay across a single link always being $\Delta_{\text{send\_max}}$. Hence, $\max(D - Heartbeat.delay)$ reduces to $(n-k)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$. This quantity will be the maximum because buffering time is accurately kept track of by the nodes and the difference $D - Heartbeat.delay$ can be increased only by having the heartbeat traverse as many links as possible.

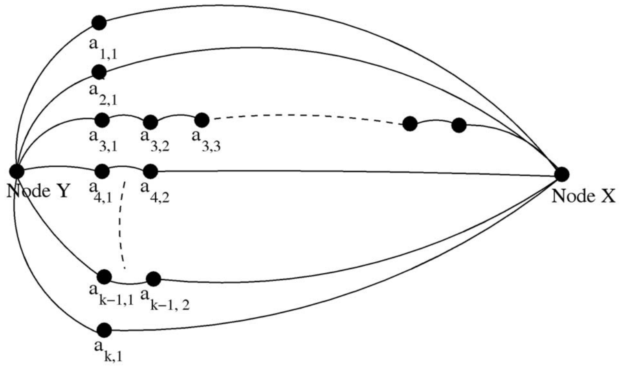


Fig. 4. Network for Proof of Lemma 9.

The above analysis holds if π is greater than $D_{\text{maxn}} - \Delta_{\text{send\_init}} - \Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}$. For smaller values of π, see [25].

Substituting for $\max(D - Heartbeat.delay)$ gives the result stated in the lemma. □

Lemma 9. *The working state holding time,* SHT_w, *is* $(d(k-3)(n-1) + 2n + k - 6)\Delta_{\text{send\_init}} + (n+k-6)\Delta_{\text{send\_max}} - (k-2)\epsilon$, *where* ϵ *is an arbitrarily small constant, if the failed state holding time,* SHT_f, *is at least* D_{maxn}.

Proof. Taking a hint from the Proof of Lemma 6, it is easy to see that the network shown in Fig. 4 will give the largest value for SHT_w. Let at time t^-, nodes $Y, a_{3,1}, a_{4,1}, \ldots, a_{k,1}$ be in the failed state. At time t, node Y recovers and node $a_{1,1}$ fails. Doing an analysis similar to the Proof for Lemma 9 gives the result stated in the lemma. The time taken for a heartbeat to reach node X since node Y's recovery is the theoretical latency in detecting a recovery event and is found to be $(d(k-2)(n-1) + n + k - 4)\ \Delta_{\text{send\_init}} + (n + k - 4)\ \Delta_{\text{send\_max}} - (k-2)\epsilon$. This quantity is less than D_{maxn}, hence the assumption $SHT_f > D_{\text{maxn}}$ is not violated. □

Lemma 10. *The heartbeat rejection time,* T_{reject}, *is* $2\rho\pi + 2\rho D_{\text{maxn}} + n(1+\rho)\ (\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}})$.

Proof. Let node Y initiate a heartbeat at real time t. Node X receives the heartbeat with delay D. Hence, node X will timeout waiting for the next heartbeat from node Y at time $t + D + (1+\rho)\pi + D_{\text{maxn}} - heartbeat.delay$ and will reject any heartbeat it may receive regarding node Y until time $t + D + (1+\rho)\pi + D_{\text{maxn}} - heartbeat.delay + (1-\rho)T_{\text{reject}}$ at the earliest. Node X could get a stale heartbeat at time $t + T_{\text{exist}}$. We require that node X still rejects heartbeats regarding node Y at this point in time. Hence, we require

$$t + \min(D - heartbeat.delay) + (1+\rho)\pi + D_{\text{maxn}} + (1-\rho) T_{\text{reject}} > t + T_{\text{exist}}.$$

Since $\min(D - heartbeat.delay) = 0$, substituting for T_{exist} and simplifying the above inequality gives the result stated in the lemma. □

Lemma 11. *The failed state holding time,* SHT_f, *is* $T_{\text{exist}} + (1+\rho)T_{\text{reject}} - D_{\text{min}} - \Delta_{\text{send\_init}}$.

Proof. The failed state holding time should be large enough so that the first heartbeat sent after a node recovers reaches other nodes after they have timed out, declared the node to be faulty and the T_{reject} timer has expired. In the worst-case, let node Y send a heartbeat at time t and fail at time $t + \Delta_{\text{send\_init}}$. Let a nonneighboring node X timeout and diagnose node Y to be faulty at time $t + T_{\text{exist}}$. Node X will then reject any heartbeat it may receive from node Y for time $(1+\rho)T_{\text{reject}}$ after this. Hence, the first heartbeat sent by node Y at time $t + \Delta_{\text{send\_init}} + SHT_f$ must reach node X after this point in time. This requirement can be expressed as

$$t + \Delta_{\text{send\_init}} + SHT_f + D_{\min} > t + T_{\text{exist}} + (1+\rho)T_{\text{reject}}.$$

Simplifying the inequality gives the result stated in the lemma. If π is assumed to be greater than $\Delta_{\text{send\_init}} + D_{\min}$ (or else the network will get flooded with heartbeats), then SHT_f will be greater than D_{maxn} which was assumed in the earlier lemmas. □

The proof for the following theorem can be found in the appendix.

Theorem 2. *With $W = 0$, a working state holding time of*

$$(d(k-3)(n-1) + 2n + k - 6)\Delta_{\text{send\_init}} + (n + k - 6)\Delta_{\text{send\_max}} - (k-2)\epsilon$$

and a failed state holding time of $T_{\text{exist}} + (1+\rho)T_{\text{reject}} - D_{\min} - \Delta_{\text{send\_init}}$, Algorithm ForwardHeartbeat achieves bounded correctness with a diagnostic latency of $T_{\text{exist}} - \Delta_{\text{send\_init}}$ and a start-up time of $(1+2\rho)T_{\text{exist}}$, if $\pi > D_{\text{maxn}} - \Delta_{\text{send\_init}} - \Delta_{\text{send\_min}} - \Delta_{\text{send\_max}}$.

If π is much greater than other parameters and n is not too large, then the working state holding time is small, and the failed state holding time and the diagnostic latency are roughly π. Thus, Algorithm ForwardHeartbeat has a diagnostic latency and state holding time of about one round.

The analysis considered thus far assumes logical heartbeat sequence numbers. In practice, due to the finite length of the field used to store the heartbeat sequence number, the maximum number that can be stored in the field (denoted by MAX_SEQ_NUM) will be reached and wrap around must occur. The following lemma defines a lower bound for MAX_SEQ_NUM, which guarantees that the algorithm will not mistake a fresh heartbeat for a stale one even when sequence numbers wrap around.

Lemma 12. *The maximum sequence number, MAX_SEQ_NUM, beyond which the sequence number wraps around to zero, must be at least $2p$, where $p = \lfloor \frac{T_{\text{exist}} - D_{\min}}{(1-\rho)\pi} \rfloor$, for Theorem 2 to hold.*

Proof. Let node Y send a heartbeat with sequence number n at real time t. Another node X will timeout at time $t + T_{\text{exist}}$ at the latest. The maximum number of heartbeats X can receive in this time is $p = \lfloor \frac{T_{\text{exist}} - D_{\min}}{(1-\rho)\pi} \rfloor$. Hence, if the sequence number of the last heartbeat received by node X is n, then heartbeats with sequence numbers $n-p$ to n should be considered as stale heartbeats, and the sequence numbers of new heartbeats subsequently received can only be between $n+1$ and $n+p$. Hence, MAX_SEQ_NUM is at least $(p-1)+1+p$, which reduces to the expression stated in the lemma. □

6 Simulation Results

Algorithm ForwardHeartbeat was simulated on randomly generated networks. $\Delta_{\text{send\_init}}$, $\Delta_{\text{send\_min}}$, $\Delta_{\text{send\_max}}$, and π were kept fixed at 0.002, 0.008, 0.08, and 60 seconds, respectively, for all simulations. Simulations were performed on networks of sizes 32, 64, 128, and 256. Simulations were done using discrete event simulation techniques. The dynamic nature of the system was modeled using a Poisson process. When an event occurs on a node, the time at which the next event occurs on the same node is the state holding time for the current state plus an additional time as given by the Poisson process. If a failure event is not possible to occur because the number of failed nodes is greater than $k-1$, then the failure event is rescheduled to a later time again according to a Poisson process. Two values for the Poisson mean were used, 1 second and 200 seconds. In all simulations, the minimum state holding times for both failed and working states were set to the failed state holding time, SHT_f.

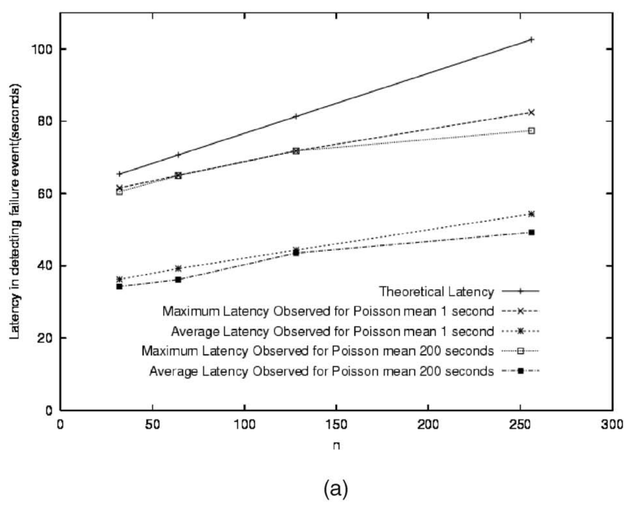
Graphs were randomly generated for a given n and k. Since every node must have at least k neighbors, links were randomly introduced such that every node has k neighbors. The connectivity of this network is then found by running the Ford-Fulkerson algorithm [10]. If the connectivity of the network is less than k, then n links are randomly introduced into the network. This process is continued until the resulting connectivity of the network is at least k. All graphs generated this way had a connectivity exactly equal to k.

Simulations were also carried out for two different values of k (3 and $\log_2 n$). To investigate the suitability of the random networks, five different networks were generated for every value of (n, k). The 95 percent confidence intervals for the average failure and recovery latencies calculated over the five random networks were found to be less than 10 percent of their respective means. Hence, we take the mean of the desired quantity yielded by the five random networks generated for a given n and k and consider it to be representative of arbitrary networks.

6.1 Effect of the Poisson Mean

Fig. 5a shows the latency in detecting failure events versus n and with $k = 3$. The Poisson mean affects the dynamic nature of the system. Two values for the Poisson mean were considered -1 second and 200 seconds. The theoretical latency is independent of the Poisson mean. Hence only a single curve for both values of the Poisson mean is shown in the figure.

For most values of n, a more dynamic fault model yields a noticeably larger latency in detecting a failure event. As seen in the theoretical analysis, the latency in detecting a failure event is variable due to the difference in the actual propagation time of the last heartbeat and the delay tracked by the intermediate nodes. Hence, the more links that are traversed by the last heartbeat that is sent by a node before it could fail, the more varied the latency will be. In a more



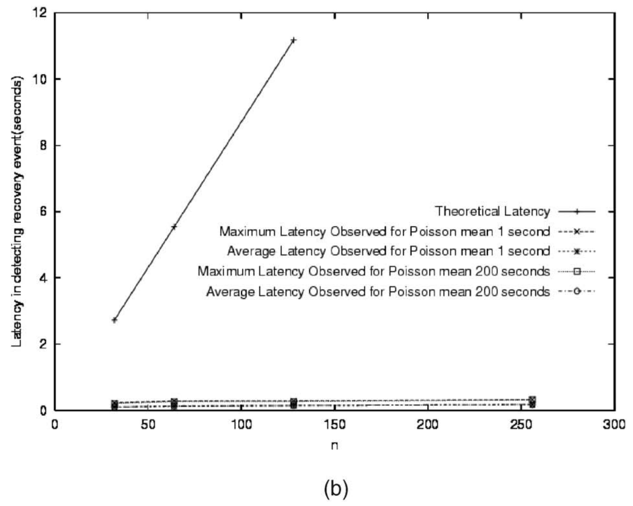


Fig. 5. Latency in detecting (a) failure events and (b) recovery events versus n for different values of the Poisson mean.

dynamic fault model, it can be expected that a heartbeat takes a longer time to propagate than in networks that have a relatively less dynamic fault model. Hence, the more dynamic fault model (Poisson mean $= 1$ second) yields a larger latency.

Fig. 5b shows the latency in detecting recovery events versus n. The theoretical latency is independent of the Poisson mean. The sequence of events considered to derive the theoretical latency yields a direct relationship with n. The observed latency shows a relatively marginal increase with increasing n. This is because, in practice, a heartbeat is sent throughout the network through multiple paths. The consequent delay does not change appreciably with increasing n.

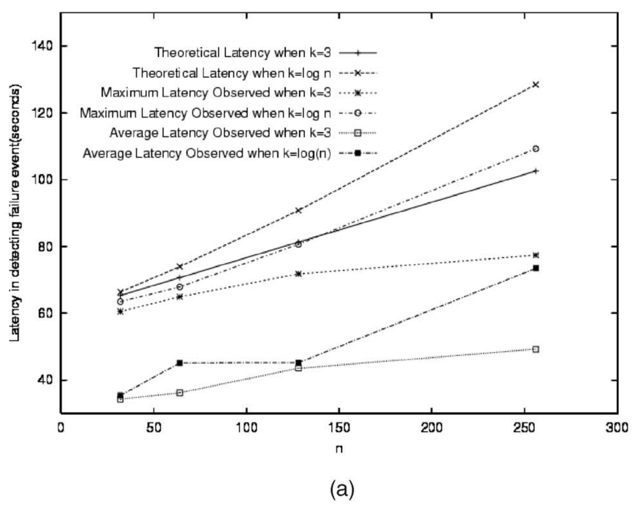
The Poisson mean seems to hardly affect the measured worst-case and average latencies in detecting recovery events. The theoretical latency for $n = 256$ is not shown in the graph because including it would scale down the y-axis such that the measured maximum and average latency curves get blurred out.

6.2 Effect of Connectivity (k)

Simulations were performed for $k = 3$ and $k = \log_2 n$, with the Poisson mean $= 200$ seconds. Higher connectivity should imply quicker propagation of heartbeats. However, in Fig. 6a, the latency becomes larger with higher connectivity. This is because the timeout period is dependent on the theoretically derived value of D_{maxn} which increases appreciably with a small increase in k. This more than offsets quicker propagation of heartbeats.

Fig. 6b shows the variation of the measured maximum and average latencies in detecting recovery events for different connectivities. The recovery latency is higher when k is lower, contradicting the theoretically expected result. With a higher k there are more links, thereby speeding propagation of heartbeats. The theory considers a specific sequence of events that yield latencies that increase with k. In practice, the occurrence of this particular sequence of events is very unlikely.

Fig. 7 shows that the number of messages per link per hearbeat period is slightly less than n for the two different



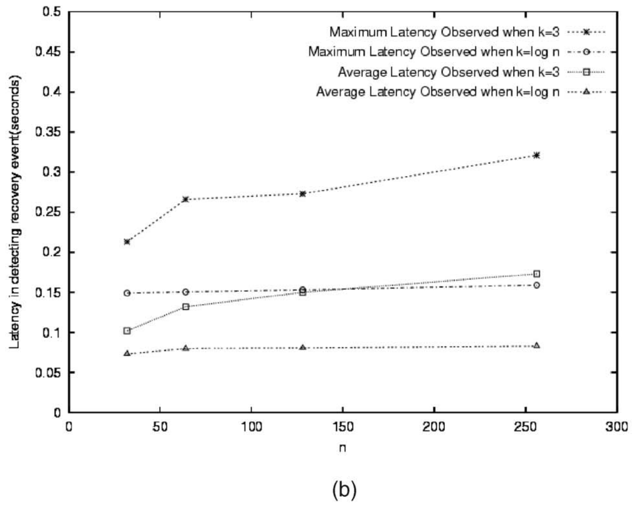


Fig. 6. Latency in detecting (a) failure events and (b) recovery events versus n for different values of connectivity.

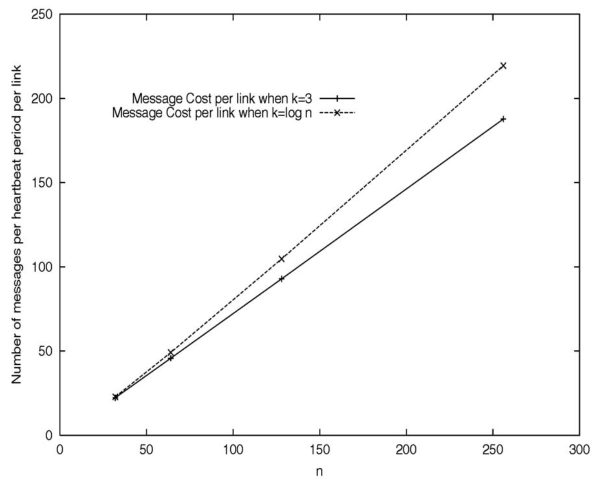


Fig. 7. Number of messages per link versus n for different values of connectivity.

connectivity values when the Poisson mean is 200 seconds. While this can be high for very large n, it is a reasonable cost for moderate n and typical heartbeat periods, particularly since the message length is 10 bytes or less in this case.

7 Discussion

The notion of bounded correctness strengthens the properties of distributed diagnosis in the presence of dynamic failures and repairs compared to existing algorithms. Bounded correctness allows arbitrary failures and/or repairs to happen as long as two consecutive state changes on a single node are not too close together in time. All nodes can change state at the same time or a cascade of status changes can occur, while maintaining a notion of correctness at all times. Algorithms HeartbeatComplete and ForwardHeartbeat were shown to handle these behaviors effectively in completely-connected and not-completely-connected networks, respectively. These behaviors are not proven to be handled correctly in any other distributed diagnosis algorithm.

Existing algorithms, particularly for not-completely-connected networks, strive to minimize the number of tests performed and hence have a difficult time handling dynamic failures. Algorithm ForwardHeartbeat propagates status information as quickly and through as many redundant paths as possible to allow it to effectively handle dynamic situations. The message complexity of ForwardHeartbeat is $O(n \cdot e)$ per hearbeat period for an n-node, e-link network. An interesting open question is whether algorithms with lower message cost can be developed that either have the same latency and state holding time as ForwardHeartbeat or allow trade offs between message cost and these quantities to be exploited.

Appendix

Proof for Theorem 1

Part 1: Bounded Diagnostic Latency

Case 1a: Fault Event. Consider the event that a node Y fails at time t and another node X is working continuously during the interval $(t, t+L)$, where L is the diagnostic latency specified in the theorem. It must be shown that X learns about Y's failure and does so by time $t+L$. The worst-case is if node Y fails immediately after sending a heartbeat to X, i.e., at time $t = t' + \Delta_{\text{send\_init}}$, where t' is the time of initiation of the heartbeat. This heartbeat arrives at X at time $t + \Delta_{\text{send\_max}}$ at the latest. At this time, node X sets its receive heartbeat timer for node Y to expire after time $(1+\rho)\Delta_{\text{heartbeat}}$.

Now, with the state holding time specified in the theorem, node Y returns to the working state later than $t + (1+3\rho)\pi + 2(1+\rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) - \Delta_{\text{send\_init}} - (1-\rho)W$. Node Y will then initiate its next heartbeat no earlier than time $(1-\rho)W$ after this. Thus, a second heartbeat from node Y to node X must arrive at X later than $t + (1+3\rho)\pi + 2(1+\rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) + \Delta_{\text{send\_min}}$. The difference in arrival times of these two heartbeats is therefore greater than $(1+3\rho)\pi + (1+2\rho)(\Delta_{\text{send\_max}} - \Delta_{\text{send\_min}}) = (1+2\rho)\Delta_{\text{heartbeat}}$. Hence, by the time the second heartbeat arrives at node X, the receive timer for node Y on node X will have already expired and node X will have been marked as failed. Hence, the event is detected.

The diagnostic latency in this case is $(1+2\rho)\Delta_{\text{heartbeat}} + \Delta_{\text{send\_max}} = (1+3\rho)\pi + 2(1+\rho)\Delta_{\text{send\_max}} - (1+2\rho)\Delta_{\text{send\_min}}$, and the latency bound is satisfied.

Case 1b: Recovery Event. Consider an arbitrary node Y recovering from a failure (transitioning to the working state) at time t. It must be shown that an arbitrary node X receives a heartbeat from node Y by time $t+L$, where L is as specified in the theorem.

With a recovery wait time of W and a state holding time at least $(1+\rho)W + \Delta_{\text{send\_init}}$, node Y will initiate a heartbeat to all of its neighbors (including node X). This heartbeat will then be received by node X by time $t + (1+\rho)W + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$. This corresponds to a latency of $(1+\rho)W + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$, which satisfies the theorem condition.

Since both types of events (failure and repair) are detected by working nodes within the latency range specified in the theorem, bounded diagnostic latency is achieved.

Part 2: Bounded Start-Up

Consider an arbitrary node X that enters the working state at time t. It must be shown that, at time $t+S$ for S equal to the start-up time specified in the theorem, node X has calculated an L-valid state for each other node. Note that $S = L$ in the theorem.

Consider an arbitrary node Y for which node X holds a particular state at time $t+S$. There are two possible ways that the start-up condition could be violated with respect to node Y. Either node X holds a state of working for node Y but Y was in the failed state during the entire interval $[t, t+S]$, or node X holds a state of failed for node Y but Y was in the working state during the entire interval $[t, t+S]$. We now show that neither of these possibilities can occur with Algorithm HeartbeatComplete.

Possibility 2a: X holds working for Y but Y was failed for entire interval. Node X can hold a state of working for node Y only if it received a heartbeat from Y during the interval $[t, t+S]$. Since Y was failed during this entire interval, the only possibility is that Y sent a heartbeat before time t and then failed, while node X received Y's heartbeat after time t and did not time out prior to time $t + S$. The latest that a heartbeat could be received by node X if node Y was failed at time t is if Y sent a heartbeat at time $t - \Delta_{\text{send\_init}}$ and then failed immediately. This heartbeat would be received by X at time $t - \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$ at the latest. At this time, X would set a timer to expire time $(1+\rho)\Delta_{\text{heartbeat}}$ later. This timer will expire at the latest by real time $t + \Delta_{\text{send\_max}} - \Delta_{\text{send\_init}} + (1+2\rho)\Delta_{\text{heartbeat}}$. Since this expression is smaller than t plus the first term in the start-up time given in the theorem, node X must time out prior to time $t + S$ and this possibility cannot occur.

Possibility 2b: X holds failed for Y but Y was working for entire interval. The shortest time interval between heartbeat intiation and a subsequent timeout is $\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}} + \Delta_{\text{heartbeat}}$. Consider the time interval $(t + S - (\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + (1+\rho)\pi), t + S - (\Delta_{\text{send\_init}} + \Delta_{\text{send\_min}})]$. Since the length of this interval is at least π time units, if node Y is continuously working during this interval it must have a sent a heartbeat in this interval. The earliest time node X can timeout after receiving a heartbeat sent in this time interval is after $t + S$. Hence, it is not possible to hold a status of $failed$ for a node at time $t + S$ for a node that has been working continuously in the interval $[t + S - (\Delta_{\text{send\_init}}\Delta_{\text{send\_max}} + (1+\rho)\pi), t + S]$.

If Y was continuously working in the interval $[t, t+S]$ but not in the interval $(t + S - (\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + (1+\rho)\,\pi), t + S]$, then Y must have recovered in the time interval $(t + S - (\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + (1+\rho)\pi), t]$. Since $S \geq (1+\rho)W + \Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$, node X will definitely receive the heartbeat sent by Y upon its recovery before time $t + S$. The earliest time X can timeout after receiving this first heartbeat is after $t + S - (\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}} + (1+\rho)\pi) + \Delta_{\text{send\_init}} + \Delta_{\text{send\_min}} + \Delta_{\text{heartbeat}}$ which is greater than $t + S$.

Hence, if node Y is in the $working$ state continuously in the interval $[t, t+S]$, then node X cannot hold a status of $failed$ for node Y at time $t + S$.

PART 3: ACCURACY

We show that, for any state transition recorded for node Y by a working node X, if the original state is valid, then the transition preserves accuracy and results in a valid next state. Since, from Part 2, the initial states held by a working node for every other node are valid at its start-up time after recovery, accuracy holds.

According to Algorithm HeartbeatComplete, a working node X changes the state of another node Y from failed to working only when it receives a heartbeat from Y and the current recorded state for Y is failed. Let the receive time of such a heartbeat message be denoted by t. If the current recorded state is a valid one, then Y was actually in the failed state at time $t - L$ or later. Now, receipt of the message implies that Y sent a heartbeat at some time in the interval $(t - \Delta_{\text{send\_max}}, t - \Delta_{\text{send\_min}})$ and was therefore working at that time. Therefore, Y must have experienced an event (transitioned from the failed state to the working state) during the interval $(t - \Delta_{\text{send\_max}} - W, t - \Delta_{\text{send\_min}} - W)$. Thus, the transition recorded by X does correspond to an actual event.

Now, consider if X changes the state of Y from working to failed. It can do this only if its receive heartbeat timer for Y expires. If the initial state for Y was valid and, therefore, Y was actually working at time $t - L$ or later, then by Lemma 1, Y must have failed prior to sending its next heartbeat. Thus, the transition corresponds to an actual event.

Multiple detections of the same event could only possibly occur if a stale heartbeat could be received after a node is timed out. However, since heartbeats are not relayed from node to node, and they are received and discarded within time $\Delta_{\text{send\_init}} + \Delta_{\text{send\_max}}$ of being sent, no stale heartbeats can exist in the system. □

PROOF FOR THEOREM 2

We restrict ourselves to the realistic case of $\pi > D_{\text{maxn}} - \Delta_{\text{send\_init}} - \Delta_{\text{send\_min}} - \Delta_{\text{send\_max}}$. For smaller values of π, see [25].

PART 1: BOUNDED DIAGNOSTIC LATENCY

Case 1a: Fault Event. We need to prove that, if node Y fails at time t, then another node X that is continuously in the working state since time t diagnoses the fault event by time $t + L$. Assume Y initiated its last heartbeat at time $t - t'$ before it could fail at time t. Then, X will timeout at time $t - t' + T_{\text{exist}}$ at the latest (see Lemma 8). We require that the heartbeat Y sends after it recovers from the fault event reaches X after X has timed out waiting for the next heartbeat from Y. This translates to proving that $t + SHT_f + D_{\min} > t - t' + T_{\text{exist}}$. Simplifying and substituting for the various quantities shows that the inequality is true. The latency in detecting the failure event is $T_{\text{exist}} - \Delta_{\text{send\_init}}$.

Case 1b: Recovery Event. We need to show that, if node Y recovers at time t, then another node X that is in the working state continuously since time t detects the recovery event by time $t + L$. A recovery event will be detected if the working state holding time, SHT_w, is large enough so that all nodes in the working state since the occurrence of the recovery event receive the heartbeat sent after the node recovered. From Lemma 9, we have that SHT_w satisfies this condition.

Next, we require that node X that has diagnosed node Y to be faulty does not reject any heartbeat it receives from node Y after node Y has recovered. In the Proof for Lemma 10, it is precisely this situation that is avoided in deriving the value of T_{reject}.

Hence, a recovery event is always detected. The latency in detecting a recovery event is smaller than the latency in detecting a failure event, thus satisfying the latency bound mentioned in the theorem.

PART 2: BOUNDED START-UP

Consider an arbitrary node X recovering at time t. By time $t + S$, it diagnoses every node in the system. We need to prove that the state diagnosed at time $t + S$ for an arbitrary node Y is L-valid. For algorithm ForwardHeartbeat, we have S equal to $(1+2\rho)T_{\text{exist}}$.

Case 2a: Status of working held at time $t + S$. We need to investigate if it is possible that node Y was in the failed state in the entire interval $[t + S - L, t + S]$. Since $S = (1+2\rho)T_{\text{exist}}$, any heartbeat initiated by node Y before time t will not exist in the network at time $t + S$. The shortest time interval between heartbeat initiation and heartbeat timeout is $\min(D + (1+\rho)\pi + D_{\text{maxn}} - Heartbeat.delay)$. With $\min\ (D - Heartbeat.delay) = 0$, this becomes $(1+\rho)\pi + D_{\text{maxn}}$. Hence, if a status of

$working$ is held by node X for node Y at time $t + S$, then node Y must have issued a heartbeat in the interval $[t + S - ((1 + \rho)\pi + D_{\mathrm{maxn}}), t + S - D_{\min}]$. Since $(1 + \rho)\pi + D_{\mathrm{maxn}} \leq L$, the $working$ state detected by the end of the startup time is L valid.

Case 2b: Status of $failed$ held at time $t + S$. From Case 2a, if node Y were to be in the $working$ state during the entire interval $[t + S - (1 + \rho)\pi - D_{\mathrm{maxn}}, t + S - D_{\min}]$, then node X will definitely hold a status of $working$ for node Y at time $t + S$. Therefore, if a status of $failed$ is held at time $t + S$, then node Y must have been in the $failed$ state sometime in the said interval. Again, since $(1 + \rho)\pi + D_{\mathrm{maxn}} \leq L$, L validity is maintained.

PART 3: ACCURACY

Let node X hold an L valid status of $working$ for node Y and detect a failure event at time t. Since $SHT_w + SHT_f > L$, there can be a maximum of only one failure event in a time interval of length L time units. If Y is continuously in the $working$ state and, hence, sends new heartbeats at its scheduled intervals of π time units, then from Lemma 7, we have the timeout value set such that X will not timeout waiting for the next heartbeat from Y, thus avoiding a false detection of a failure event. Hence, a failure event that is detected corresponds to an actual failure event that occurred on Y in the time interval $[t - L, t]$ and does not correspond to any other failure event that had occurred on Y. The resulting $failed$ state detected is again L valid.

Next, let node X hold an L valid status of $failed$ for node Y at time t. X detects a recovery event on Y by the arrival of a heartbeat at time t. From Lemma 10, X rejects any heartbeat it may receive from Y for sufficient time such that no more heartbeats from the failed node Y can be received after the T_{reject} phase. Hence, this heartbeat must have been sent by Y in the interval $[t - D_{\mathrm{maxn}}, t - D_{\min}]$. Since the initial view of $failed$ is L valid, a recovery event indeed occurred on Y. Since $SHT_w + SHT_f > L$, a maximum of one recovery event can occur in a period of length L time units. Hence, the detected recovery event corresponds to this particular recovery event that occurred on Y in the interval $[t - L, t]$ and not to any other recovery event that might have occurred earlier on Y. The resulting $working$ state detected is again L valid. □

ACKNOWLEDGMENTS

This research was supported by the US National Science Foundation under Grant CCR-9803741.

REFERENCES

[1] M.K. Aguilera, W. Chen, and S. Toueg, "Failure Detection and Consensus in the Crash-Recovery Model," *Distributed Computing,* vol. 13, no. 2, pp. 99-125, 2000.

[2] A. Bagchi and S.L. Hakimi, "An Optimal Algorithm for Distributed System Level Diagnosis," *Proc. Digest of the 21st Int'l Symp. Fault Tolerant Computing,* pp. 214-221, 1991.

[3] M. Barborak, M. Malek, and A.T. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys,* vol. 25, pp. 171-220, June 1993.

[4] D. Blough and H. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Trans. Computers,* vol. 48, pp. 470-493, May 1999.

[5] R. Bianchini and R. Buskens, "Implementation of On-Line Distributed System-Level Diagnosis Theory," *IEEE Trans. Computers,* vol. 41, pp. 616-626, May 1992.

[6] M. Clegg and K. Marzullo, "A Low-Cost Processor Group Membership Protocol for a Hard Real-Time Distributed System," *Proc. 18th Real-Time Systems Symp.,* pp. 90-98, 1997.

[7] F. Cristian, "Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems," *Distributed Computing,* pp. 175-187, 1991.

[8] E.P. Duarte Jr., A. Brawerman, and L.C.P. Albini, "An Algorithm for Distributed Hierarchical Diagnosis of Dynamic Fault and Repair Events," *Proc. Seventh Int'l Conf. Parallel and Distributed Systems,* pp. 299-306, 2000.

[9] E.P. Duarte Jr. and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," *IEEE Trans. Computers,* vol. 47, pp. 34-45, Jan. 1998.

[10] S. Even, *Graph Algorithms.* Computer Science Press, 1979.

[11] P. Ezhilchelvan and R. de Lemos, "A Robust Group Membership Algorithm for Distributed Real-Time Systems," *Proc. 11th Real-Time Systems Symp.,* pp. 173-179, 1990.

[12] M. Hiltunen, "Membership and System Diagnosis," *Proc. 14th Symp. Reliable Distributed Systems,* pp. 208-217, 1995.

[13] S. Hosseini, J. Kuhl, and S. Reddy, "A Diagnosis Algorithm for Distributed Comp. Systems with Dynamic Failure and Repair," *IEEE Trans. Computers,* vol. 33, pp. 223-233, Mar. 1984.

[14] W. Hurwood, "Ongoing Fault Diagnosis," *Proc. 15th Symp. Reliable Distributed Systems,* pp. 108-117, 1996.

[15] K.H. Kim et al., "An Efficient Decentralized Approach to Processor-Group Membership Maintenance in Real-Time LAN Systems: The PRHB/ED Scheme," *Proc. 11th Symp. Real-Time Distributed Systems,* pp. 74-83, 1992.

[16] H. Kopetz and G. Grunsteidl, "TTP—A Protocol for Fault-Tolerant Real-Time Systems," *Computer,* vol. 27, no. 1, pp. 14-23, Jan. 1994.

[17] P. Maestrini and P. Santi, "Self Diagnosis of Processor Arrays Using a Comparison Model," *Proc. 14th Symp. Reliable Distributed Systems,* pp. 218-228, 1995.

[18] A. Pelc, "Undirected Graph Models for System-Level Fault Diagnosis," *IEEE Trans. Computers,* vol. 40, pp. 1271-1276, Nov. 1991.

[19] S. Rangarajan, A.T. Dahbura, and E. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Trans. Computers,* vol. 44, pp. 312-334, Feb. 1995.

[20] S. Rangarajan and D. Fussell, "Diagnosing Arbitrarily Connected Parallel Computers with High Probability," *IEEE Trans. Computers,* vol. 41, pp. 606-615, May 1992.

[21] J. Rushby, "A Formally Verified Algorithm for Clock Synchronization under a Hybrid Fault Model," *Proc. 13th ACM Symp. Principles of Distributed Computing,* pp. 304-313, 1994.

[22] R.D. Schlichting and F.B. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Trans. Computer Systems,* vol. 1, no. 3, pp. 222-238, Aug. 1983.

[23] A. Sengupta and A.T. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," *IEEE Trans. Computers,* vol. 41, pp. 1386-1396, Nov. 1992.

[24] M. Stahl, R. Buskens, and R. Bianchini, "On-Line Diagnosis in General Topology Networks," *Proc. Workshop Fault Tolerant Parallel and Distributed Systems,* 1992.

[25] A. Subbiah, "Design and Evaluation of a Distributed Diagnosis Algorithm for Arbitrary Network Topologies in Dynamic Fault Environments," MS thesis, Georgia Inst. of Technology, http://www.ece.gatech.edu/~arun/ms_thesis.pdf, 2001.

[26] P.M. Thambidurai and Y.K. Park, "Interactive Consistency with Multiple Failure Modes," *Proc. Seventh Symp. Reliable Distributed Systems,* pp. 93-100, Oct. 1988.

Arun Subbiah received the BTech degree in electrical engineering from the Indian Institute of Technology, Madras, in 2000 and the MS degree in electrical and computer engineering from the Georgia Institute of Technology in 2001, where he is currently a PhD student. His research interests are fault tolerant computing, distributed systems, and security. He is a student member of the IEEE and the IEEE Computer Society.

Douglas M. Blough received the BS degree in electrical engineering and the MS and PhD degrees in computer science from The Johns Hopkins University, Baltimore, Maryland, in 1984, 1986, and 1988, respectively. Since the Fall of 1999, he has been a professor of electrical and computer engineering at the Georgia Institute of Technology, where he also holds a joint appointment in the College of Computing. From 1988 to 1999, he was with the faculty of electrical and computer engineering at the University of California, Irvine. Dr. Blough was program cochair for the 2000 International Conference on Dependable Systems and Networks (DSN) and the 1995 Pacific Rim International Symposium on Fault-Tolerant Systems. He has been on the program committees of numerous other conferences, was associate editor of *IEEE Transactions on Computers* from 1995 through 2000, and is currently an associate editor of *IEEE Transactions on Parallel and Distributed Systems*. His research interests include distributed systems, dependability and security, and wireless ad hoc networks. He is a senior member of the IEEE and the IEEE Computer Society.