Scalable, Adaptive, Time-Bounded Node Failure Detection *

Matthew Gillen, Kurt Rohloff†, Prakash Manghwani, Richard Schantz
BBN Technologies
10 Moulton St., Cambridge, MA 02138, USA
{mgillen, krohloff, pmanghwani, schantz}@bbn.com

Abstract

This paper presents a scalable, adaptive and time-bounded general approach to assure reliable, real-time Node-Failure Detection (NFD) for large-scale, high load networks comprised of Commercial Off-The-Shelf (COTS) hardware and software. Nodes in the network are independent processors which may unpredictably fail either temporarily or permanently. We present a generalizable, multi-layer, dynamically adaptive monitoring approach to NFD where a small, designated subset of the nodes are communicated information about node failures. This subset of nodes are notified of node failures in the network within an interval of time after the failures. Except under conditions of massive system failure, the NFD system has a zero false negative rate (failures are always detected with in a finite amount of time after failure) by design. The NFD system continually adjusts to decrease the false alarm rate as false alarms are detected. The NFD design utilizes nodes that transmit, within a given locality, "heartbeat" messages to indicate that the node is still alive. We intend for the NFD system to be deployed on nodes using commodity (i.e. not hard-real-time) operating systems that do not provide strict guarantees on the scheduling of the NFD processes. We show through experimental deployments of the design, the variations in the scheduling of heartbeat messages can cause large variations in the false-positive notification behavior of the NFD subsystem. We present a per-node adaptive enhancement of the NFD subsystem that dynamically adapts to provide run-time assurance of low false-alarm rates with respect to past observations of heartbeat scheduling variations while providing finite node-failure detection delays. We show through experimentation that this NFD subsystem is highly scalable and uses low resource overhead.

*This work was supported by the Defense Research Projects Agency (DARPA) under contract NBCHC030119.

†Corresponding Author.

1 Introduction

Due to the decrease in cost of COTS hardware, there has been a surge in the use of highly-interconnected large-scale distributed computing environments [9, 11, 12] to host mission-critical systems. In order to maintain constant availability, automated resource managers are used in these systems to allocate computational resources on nodes in these networks to specific mission-critical applications and processes. If a resource manager can be notified of node failures by a Node Failure Detection (NFD) subsystem in a fast, timely manner with a low false-alarm rate, it is feasible to redeploy mission-critical applications from the failed nodes onto auxiliary nodes within the bounds of the application's real-time deadlines to mask or ameliorate the failure. This fast, low-error notification of node failures is often a key component of a resource management system's ability to provide the constant availability of mission-critical applications in real-world distributed computing environments. However, to be easily adoptable and usable in real-world environments, any general designs for fast NFD subsystems must be scalable to large systems. Additionally, NFD subsystems must be low-overhead so that they will not adversely impact the operations of the underlying environment. This motivates the need to design a general purpose, scalable, low-overhead, fast and low-false alarm rate NFD subsystem.

A simple approach to NFD is Heartbeat Failure Detection (HBFD) where the nodes periodically send short "NodeAlive" heartbeat messages to the resource managers to signify that the node is still operating [8]. The node is said to "timeout" if the resource manager goes too long without observing a heartbeat message from it. In the case of a timeout, the resource manager could declare the node that timed-out had failed and then take whatever actions are necessary to redeploy mission-critical applications that were running the failed node and avoid deploying new applications on that unavailable node. Unfortunately, many potential implementations of HBFD are not usable at the scale of thousands of nodes either because of a single point

of failure when all heartbeats go to a single node; this would also cause very high overhead for the node that received all those heartbeats or a design that uses a fully connected graph (i.e. each node sends heartbeats to every other node). Additionally, previous approaches to HBFD have not been able to dynamically adapt to the run-time behavior of the system to maintain an optimal balance among various real-time operational constraints such as the node-failure detection delay and false-alarm rate.

Low-cost environments comprised of COTS hardware and software, as is frequently the case in many real-world distributed computing environments, are commonly limited to soft-real-time behavior. Soft real-time systems take a best-effort approach to scheduling computations and communications, and this is the typical situation when using commodity operating systems [7]. As a result, there is always variation in the scheduling of heartbeat transmissions generated by the nodes in soft-real-time systems. These variations in the scheduling of heartbeat messages can cause large variations in the false-positive notification behavior of the NFD subsystem.

The amount of variation in heartbeat scheduling varies greatly from node to node in NFD subsystems. In practice, one could manually tune the behavior of an NFD subsystem to account for the node-by-node variations in heartbeat scheduling but this is a labor-intensive process and is not feasible in very large computing environments with large numbers of nodes when cost is a factor. Additionally, no amount of manual tuning may actually solve the problem as any changes in system behavior or use may require retuning. This inability to manually tune an NFD subsystem may cause the NFD system to repeatedly and falsely declare large numbers of node failures due to the variations in heartbeat generation times. Although not as disruptive as false negatives, false positive node failures are highly undesirable because if a node is incorrectly declared failed, the resource manager may unnecessarily deploy mission-critical applications onto other nodes. This might cause the unnecessary interruption of service for the mission-critical applications, in addition to the depletion of available nodes.

In this paper we present a real-time dynamic, adaptive heart-beat methods for node failure detection that can be used to achieve a low false-alarm rate. Our design goal is to have a dynamic, adaptive NFD system that assures a low false-alarm rate at the cost of a potentially higher (but still finite) node-failure detection delay. The system should be able to assure a very small false alarm rate if future behavior is similar to past observed behavior. These design concerns for NFD have not been previously discussed in the literature. We show how to enhance this NFD subsystem so it can quickly adapt to decrease false-positive rates on a per-node basis for variations in scheduling due to a system's soft-real-time components. This adaptation trades off potential increases in (a time-bounded) worst-case failure notification time for assured limitations on false-positive notification rate (based on past behavior). We have the additional design concern that the NFD system should be low-overhead and scalable to large systems and could also be easily dropped in as a common off-the-shelf service into an ongoing and enduring system management design.

There is a large and active literature associated with NFD [2–6]. However, this literature is largely focused on theoretical results and there has been insufficient real-world development and experimentation with general approaches to scalable, adaptable NFD subsystems for distributed computing environments comprised of COTS hardware and software. The adaptive, hierarchical NFD design and the experimentation is motivated by the DARPA ARMS program [9, 10] which required a generalizable, scalable and fast NFD system with low overhead that could satisfy the conflicting requirements of fast node failure detection and low false alarm rates.

This paper is organized as follows. In Section 2 we discuss the specific requirements of the NFD system. In Section 3 we present the NFD design. In Section 4 we discuss experiments with a non-adaptive version of the design. In Section 5 we describe an adaptive version of the NFD system. A set of conclusions from the work is discussed in Section 6.

2 System Architecture and NFD Design Requirements

As discussed in the introduction, we assume that the resource manager (or, for the sake of generality, whatever component in the distributed computing environment maintains information about node availability) maintains a Node Status Receiver (NSR) process to track which of the nodes has failed based on signals from the NFD system. The nodes in the network run commodity, multi-threaded soft-real-time operating systems which take a best-effort approach to event scheduling. This best-effort approach to event scheduling often leads to frequent false alarms which could potentially be catastrophic for the behavior of the overall system.

To facilitate the heartbeat approach to NFD, each node maintains a thread that schedules the transmission of heartbeat messages and other operations. Generally, variations in a node's scheduling of heartbeats is dependent upon the node's processor load and several other variables, such as the degree to which kernel actions are preemptable. Although variations in scheduling from heartbeat-to-heartbeat are not necessarily deterministic to an external observer, scheduling errors from interval to interval are generally correlated.

The distributed computing environment's underlying

communication network, which is used to move heartbeats and other messages between nodes, is assumed to be a standard asynchronous data network, such as a standard IP network. Thus, it can be assumed that when the network is under a heavy operational load, there may be congestion and delayed or dropped heartbeats which could cause false positive node failure detections.

When a node is operating and connected to the communication network, it is considered alive. A node is considered dead if it has a hardware failure that prevents computation from proceeding or there is a network partition such that the node is functional but it is separated from the network (implying that it is effectively dead for the purpose of hosting parts of distributed applications) or any other intermediate condition preventing the heartbeats from reaching its destination. Possible causes of node death include the physical failure of the node, an OS crash, and the failure of the node's communication link(s).

Ideally, the NSR would be informed of node failures as soon as possible after failure with a low false positive rate. Also the NFD would ideally have low overhead, be scalable to very large computation networks and be tolerant of NFD component failures. We describe these requirements as follows:

- *Low False Positive Rate* - To prevent the resource manager from unnecessarily redeploying mission critical applications, the NFD solution should not generate erroneous failure notifications (false-positives). Repeated false positives are particularly undesirable due to the possibility of causing the resource manager to "thrash" when (re)deploying to "thrash" when (re)deploying computational processes on nodes. A goal for an NFD subsystem design beyond the current state of the art in the context of the other requirements was to assure at most one false positive node failure detection per month of operation (based on past behavior).

- *Fast Worst-Case Detection Time* - There is a need for a general, off-the-shelf deployable NFD subsystem with worst-case detection times as fast as possible with respect to need for a low false positive rate. Based on previous experience in developing NFD subsystems, we perceived that a worst-case detection time on the order of 100 milliseconds is fast enough to support soft-real-time deadlines of many applications, yet it is still beyond the ability of current scalable, low-overhead NFD subsystems with assured low false positive rates for distributed computing environments comprised of COTS components [10].

- *Low Overhead* - The NFD will need to run concurrently (i.e., on the same nodes) with mission-critical applications. Therefore, the NFD processes that run on each node of the network need to be low overhead in terms of CPU utilization. In order to be usable in a high-demand, high-load environment an NFD subsystem should require no more than 2% of the bandwidth and CPU time from any shared nodes.

- *Scalability* - The NFD solution needs to be scalable to both small and very large configurations. Our design goal was an NFD subsystem that could scale to a system with 1000's of nodes.

- *Fault Tolerance* - The failure of any one part of the NFD subsystem should not cause the failure of the entire NFD subsystem. That is, there should be no Single Point Of Failure (SPOF).

Taken individually, each requirement we place on the general NFD subsystem design may have well-known solutions in the NFD literature [4–6]. However, the combination of all these requirements in a real-world context, particularly scalability, a 100ms desired detection time and SPOF fault-tolerance make this a difficult and open problem.

3 Node Failure Detector Design

We developed a general design for an NFD subsystem comprised of a three-level hierarchical architecture. There are three component levels in this hierarchy: Node Status Receiver (NSR), Monitor, and Sender (each component may contain multiple threads of execution.) In order to satisfy the fault tolerance requirement, it is assumed that there are at least two NSR processes running on separate nodes in the system that collect aggregate information about node failures.

At the lowest level of the hierarchy, the Sender is a single thread that executes on each node of the system to generate heartbeats that are sent to a subset of the Monitors. The set of all Senders that send heartbeats to a particular monitor are called a cluster. There are multiple Monitors in the system, deployed on a small subset of the nodes. Each Monitor is comprised of two threads of execution. One thread, called the detection thread, observes the arrival of heartbeats from Senders assigned to it and saves statistical information related to its observations of heartbeats. The second Monitor thread, called the Sweeper thread, periodically executes to determine which nodes may be dead and sends lists of the nodes that the Monitor believes to be dead to the NSRs. A schematic of communication for the implementation of this NFD system can be seen in Figure 1 where all Senders in a cluster are assigned to the same two monitors (although this is not a requirement.)

The heartbeat messages are UDP packets that contain information that identifies the node that generated the heartbeat and a sequence number so that missed heartbeats can

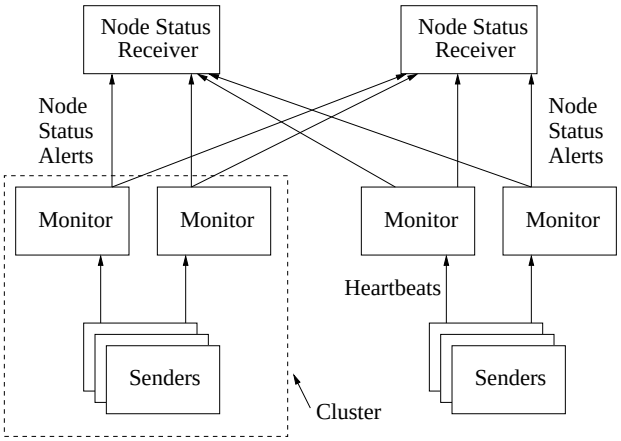

Figure 1. NFD Communication Schematic.

be detected. The Sender thread generates heartbeats which are sent to the Sender's Monitors at a nominal configurable rate; we say the heartbeats are generated at a nominal rate because we assume that there may be variations in the scheduling of this heartbeat generation thread due to the effects of using a commodity operating system.

Senders are clustered based on physical proximity in order to minimize the communication overhead associated with NFD operation. In order to satisfy the minimum requirement for fault tolerance, every Sender should be associated with at least two Monitors so that the failure of any one Monitor will not hinder the NSRs' notification of individual node failures.

The Monitor's detection thread is run at the second-highest priority, behind the sweeper thread. The monitor maintains an internal list of the arrival times of the latest heartbeat observed from each of its senders. Upon the arrival of heartbeat messages, the detection thread is scheduled to start as soon as possible, which then places a timestamp for the heartbeat on the internal list at the location corresponding to the Sender that generated the heartbeat.

The Monitor's sweeper thread is run at the highest priority the operating system has. This thread is run periodically at a nominal frequency to determine which of the nodes could be dead based on the list of the last observed heartbeat arrival times. Again, we say nominal due to the use of soft real-time operating systems which may cause variations in the scheduling of the sweeper thread. However, because this thread is run at the highest priority, there would generally be less variation in sweeper scheduling than in the scheduling of heartbeat variation.

To determine if a node has failed, the Monitor maintains a local, configurable list of Detection Threshold (Th) values for each node in its cluster. During each iteration of operation, the sweeper thread loops through a list of all the nodes that it is monitoring to detect node timeouts. If the difference between the current time and the time of the last heartbeat received by the node is greater than the Th value assigned to the node by the Monitor, the node is declared dead.

We concluded that the best practice for avoiding a large number of false positives due to dropped heartbeat packets is to set Th to be at least twice the heartbeat generation period. This allows the system to avoid declaring an erroneous failure in the case of (non-consecutive) single-packet losses without incurring the overhead from sending more packets.

In the Appendix, we develop an expression for the maximum amount of time it takes a Monitor in the NFD subsystem to send an NSR information about a node failure. We find this maximum failure notification time to be:

$$SM + MN + Th + SI + 3SL$$

where:

- SM is the communication latency of the last heartbeat sent to the Monitor from the failed node.
- MN is the worst-case communication latency of the failure notification sent from the Monitor to each of the NSR instances.
- Th is the timeout threshold used by the Monitor to detect the node failure.
- SI is the period between executions of the sweeper thread in the Monitor.
- SL is the amount of time the sweeper thread takes to run.

Generally, there will be variations in the values of SM, MN, SI and SL. However, in experimentation, we found that the values of these parameters are tightly distributed around expected median values. This provides us with a high level assurance that failure notifications will be sent to the NSRs within some time bound.

We implemented the NFD subsystem for several sets of experiments using the ARMS configuration as the context for evaluation. First we consider the case of fixed thresholds in the next section and the case of variable threshold in the section following that.

4 Fixed Threshold Experimentation

We ran two experiments of the NFD implementation using a fixed detection threshold. In the first experiment, we configured the NFD subsystem using 40 physical nodes to simulate 1020 monitored nodes. This experiment was designed to test the scalability and measure the overhead and

node failure detection times of the subsystem design when there are large numbers of node failures.

For the second experiment, we configured the NFD subsystem using 10 physical nodes (and no virtual nodes) in a high-load environment running 10 Sender instances, 2 Monitor instances and 2 NSR instances. The intent of the second experiment was to observe the effects of scheduling errors in the Senders and Monitors and to test the ability to detect false alarms.

4.1 First Experiment

We based the experiments on the motivating ARMS distributed computing environment which uses a standard IP over Ethernet network with very high bandwidth. For this experiment, we used 20 (of the 40 total) physical nodes to each run 50 instances of the Sender, providing 1000 simulated nodes. We used the remaining 20 physical nodes to run one instance each of a Sender, a Monitor. Two of these physical nodes also ran an NSR instance. Thus, we used 20 monitors, and given that we used cluster sizes of approximately 100, we used 2 Monitors for each cluster. The heartbeat generation rate was set to 45Hz, and the Monitor sweeper rate was set to 40Hz. (This implies SI, the wakeup interval for the sweeper, was 25ms.) Based on an objective of 100ms worst-case detection time, we set Th to be 50ms.

For this experiment, we used the University of Utah's Emulab [1] testbed to emulate the virtual nodes. This experimental testbed allows users to configure VLAN-based private networks and specify the operating systems emulated on the virtual nodes. We used the "pc3000" option [1] to specify the hardware on all virtual nodes. (This options specifies that the virtual nodes emulate a 3.0 GHz 64-bit Xeon processors, based on Dell Poweredge 2850.) We also specified Fedora Core 4 as the operating system on the virtual nodes. The VLAN-based network topology was just a single LAN with all 44 physical nodes on the same subnet. The nodes were unloaded with other processes.

We performed experiments to determine the runtime of the sweeper thread (SL). In the worst case when using a cluster size of 100, this thread runs in 2ms. The runtime was proportional to the number of failures detected; the more failures, the higher the runtime. The high value of 2ms was for 100 simultaneously-induced node failures.

We never observed SM or MN to be greater than 4ms. With these maximum values, $SM + MN + Th + SI + 3SL$ would be less than 91ms, which would satisfy the requirement that the NSRs would receive notifications about node failure within 100ms. This assertion is supported by experimentation. When we ran this experimental configuration, we induced over 4800 virtual and physical node failures. Over the 4800 induced node failures, all node failures were reported within the required 100ms deadline, which means there were no false negative node failure occurrences.

Based on the initial experimentation, the design scaled sufficiently for the test system with 1000 virtual nodes. In general, the NFD subsystem implementation performed with low overhead in the large-scale test system, both in terms of computation and communication resources. Additionally, the NFD subsystem implementation demonstrated its designed fault tolerance property when we induced failures in individual components such as the Monitors and NSRs, the overall behavior of the system was not altered. The virtual nodes in the system had a light computational load so the false positive rates were not tested in this experiment.

4.2 Second Experiment

In a second experiment, we ran an ARMS configuration of the NFD subsystem in a distributed computing environment comprised of 10 Senders on 10 physical nodes, along with a Monitors and an NSR coresident on two of those resident on two of those nodes. We used a different experiment testbed for this experiment because experiments on Emulab are normally limited to 16 hours and we wanted to observe long-term false positive node failure detection rates way beyond that limit.

For these experiments we used 10 workstations and servers which would normally be found in a general office environment. The mix of hardware consisted of PC desktops (Dell OptiPlex GX270 w/ Pentium-4 CPU 2.80GHz , Dell OptiPlex GX150 w/ Pentium-III CPU 933MHz and Dell PowerEdge 650 w/Pentium-4 CPU 2.66GHz) running various flavors of the non-real-time Linux operating systems (Fedora Core 5, Ubuntu 6.10, Gentoo.) This heterogeneity was intended to represent the more normal case where uniformity of hardware and software cannot be guaranteed.

The desktops used were active developer workstations, so there was typically a heavy but irregular load on these machines and on the network interconnecting them during normal business hours. We observed false positive rates that varied from one a week to several times an hour. This false alarm rate is unacceptably high; several orders of magnitude higher than desired. After analysis of the experimental runs, we concluded that there were two primary causes of false positives: dropped heartbeat packets and the inconsistent scheduling of the sender and sweeper threads.

Although it was common for single UDP heartbeat packets to be dropped in the communication network in the test system, it was sufficiently unusual for consecutive packets to be dropped when a node was still alive that we never saw a false positive caused by two consecutive node losses in any of the experiments.

We observed a large number of false alarms due to

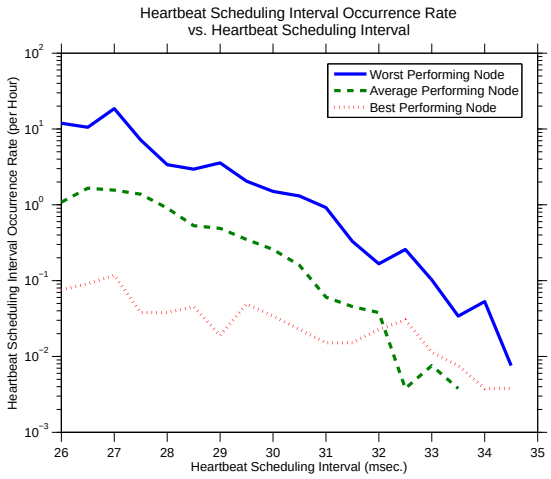

Figure 2. Plots of How the Rate of Scheduling Violation Occurrences Varies with the Scheduling Interval for the "Best", the "Worst" and an "Average" Performing Nodes on a Semi-Log Scale.

scheduling variations at the Senders and Monitors. False positives due to Monitor scheduling variations were easy to detect and compensate for during system operation. The sweeper thread has the highest priority in the Monitor so that it could preempt the operation of the detection thread. If a scheduling problem occurred that prevented the sweeper thread from executing, then it is guaranteed that the detection thread hasn't been able to execute either, and therefore heartbeats are very likely to be sitting in the Operating System's network stack. This would lead to the Monitor determining that all nodes in its cluster were failed, but most likely these "failures" would be false-positives. We implemented checks in the sweeper to detect when its scheduling has been interrupted, and if it has then it skips the failure detection until the next period. This results in the Sweeper being able to reject all false positives due to sweeper scheduling errors during our experimentation. Although this design may result in the delayed detection of real node failures during sweeper scheduling errors, we observed during our experimentation that the probability of these events happening coincidently are sufficiently low that this should not be a concern, especially with redundant monitors.

The other cause of false positives, scheduling variations at the Senders, is not as easy to detect in real-time. We observed in the experiments that these variations could prevent the Sender from generating its heartbeat message for up to a second after it nominally should have been sent.

Figure 2 contains graphs of rates of scheduling violation occurrences varies with the scheduling interval for three different nodes (on a semi-log scale). Note that we only plotted the rate of violation occurrences when the scheduling interval is greater than the nominal value of 25 msec. because by definition a scheduling violation has not occurred if the scheduling interval is less than 25 msec.. We selected the nodes out the ten available to demonstrate a) the number of observed scheduling violations exhibited by the "best" performing node (the dotted line), b) the number of observed scheduling violations exhibited by an "average" node (the dashed line) and c) the number of scheduling violations exhibited by the "worst" performing node (the solid line.) Note that from these plots we can surmise that all of the nodes exhibited scheduling violations. As can be seen in Figure 2, there is a variability in the number and magnitude of heartbeat scheduling violations of several orders of magnitude between the "best" node and the "worst" node. One could diminish these consistently occurring scheduling violations by appropriately tuning the static Monitor thresholds by hand, but this is not feasible for very large scale systems as discussed in the introduction. In the next section, we describe an extension to the design to include an adaptive NFD Monitor that can self-tune for these scheduling violations in order to trade off an improved false alarm rate for a slight increase in detection time.

5 Adaptive NFD

We designed an adaptive version of the NFD subsystem where the failure-detection thresholds used by individual Monitors are adjusted (in a strictly increasing manner) on a per-node basis. We took a simplistic approach to Monitor adaptation by multiplying the Monitor's detection threshold Th for a node by a configurable value k every time a false positive is detected on that node by the Monitor. (The same value of k is used for all nodes.) A false positive is detected when a Monitor receives a heartbeat from a node after that node has timed out and the Monitor has communicated to an NSR that the node is dead. Although this approach does not predict the occurrence of false alarms before they occur (this is most likely impossible), this approach prevents false alarms from occurring if the scheduling errors exhibited by the nodes in the future is comparable to the scheduling errors exhibited by the nodes in the past.

This approach to NFD adaptation allows the NFD software to "discover" appropriate thresholds for each individual node to avoid repeated false-positives in the future. This method also keeps stricter bounds on the worst-case detection time of nodes that are able to accurately schedule the generation of heartbeats. An additional benefit of this approach is that the threshold adjustments can be propagated to the NSR (top-tier) nodes, so that resource-management software that consumes node failure events will be able to

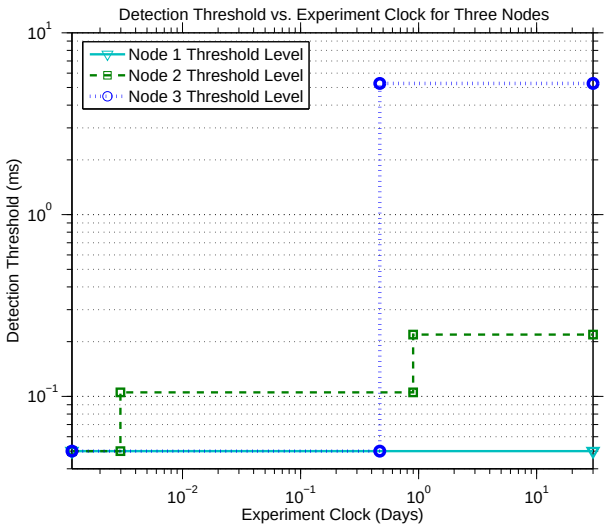

Figure 3. Sample Variations in Detection Thresholds.

take worst-case node failure detection times and false positive rates into account when deploying mission-critical applications or applications that would be difficult to redeploy.

We configured and evaluated our adaptive NFD test system in the same high-load, 10 node network from Subsection 4.2 with the threshold growth parameter k set to 2. We ran the system for a month (30 days). A sample of how the detection threshold was adjusted over the first few days of the experimentation period for three sample nodes can be seen in Figure 3. (Note that Figure 3 is plotted on a semi-log scale.) We selected the three nodes to demonstrate the highest threshold adjustment exhibited during the experimental run (Node 3), the lowest threshold adjustment exhibited during the experimental run (Node 1) and the threshold adjustments of an "average" case node (Node 2.)

For most nodes there was only one false positive, and hence one adjustment of the threshold for the entire 1-month run of the system. Note that the best performing node had no adjustments during the test run. The worst performing node only had one threshold adjustment, but this adjustment was driven by a large delay in a heartbeat arrival that was most likely an uncommon event as we did not observe any other heartbeat arrival delays of this magnitude during the experiment. "Average node" adjustments, like the one seen in Figure 3 occurred sometime after the first few minutes of operation but before the first day. No node exhibited more than 2 adjustments. This demonstrates that despite using a simplistic threshold adjustment scheme, our system was able to properly account for possible future false positives based on past observations and avoid thrashing in the system.

One might expect that this method for threshold adjustment would increase the detection threshold at an exponential rate. However, the occurrence of scheduling errors was found to be approximately exponentially distributed. This behavior causes the detection threshold to increases quickly during the tuning phase in the first few hours of operation. Later, the rate of increase of the detection threshold decays quickly, causing stable behavior in later stages.

Note that the threshold for Node 3 increased to nearly 5 seconds. A threshold this high may not be usable for most applications and suggests that an area for continued research and development would be in policies that would remove nodes from operation if they cannot reliable provide sufficient detection thresholds and false alarm rates.

6 Conclusion and Discussions

We developed a fast, general, reliable and scalable node failure detection subsystem using a hierarchical, adaptive approach. Building upon a proof-of-concept implementation, we were able to increase the timeliness of NFD in the system by making the sending and receiving of heartbeats more efficient and less susceptible to load on the host system, both on a per-host basis as well as in the overall system. Through experimentation we demonstrated how this design was enhanced to adaptively trade failure detection time for a decrease in the subsystem's false positive rate.

A design augmentation approach for future work to decrease the NFD overhead would be to make the Senders' heartbeat transmission rate dependent on changes in its detection threshold value Th at the Monitors. This is a possible area of application of variable controls rate, a current research topic in control theory for adapting control sensing and actuation rates. An underlying goal of the work in this research area is the need for methods to design these dynamic systems such that their behavior is predictable and certifiable.

References

[1] http://www.emulab.net.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, Apr. 2000.

[3] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(5):561–580, 2002.

[5] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. *2001 Pacific Rim International Symposium on Dependable Computing*, 2001.

[6] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, 2002.

[7] M. Jones and J. Regehr. Issues in using commodity operating systems for time-dependent tasks: Experiences from a study of windows nt. In *Proceedings of the Eighth Internations Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV–1998)*, pages 107–110, July 1998.

[8] M. Reynal. A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News*, 36(1):53–70, 2005.

[9] K. Rohloff, Y. Gabay, J. Ye, and R. Schantz. Scalable, distributed, dynamic resource management for the ARMS distributed real-time embedded system. In *Proceedings of the International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS-2007)*, 2007.

[10] P. Rubel, J. Loyall, R. Schantz, and M. Gillen. Fault tolerance in a multi-layered DRE system: A case study. *Journal of Computers*, 1(6):43–52, 2006.

[11] V. Savant, S. Papavassiliou, J. Tupino, and A. Zawadzki. Enhanced network management for online services. In *Proceedings of the Seventh International Conference on Computer Communications and Networks (ICCCN-1998)*, 1998.

[12] P. Verissimo and A. Casimiro. The timely computing base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930, Aug. 2002.

A An Expression for Node Failure Notification Time

In this appendix, we derive an expression for the maximum amount of time it takes for an NSR to be notified of node failures based on network latency, heartbeat generation frequency, the frequency of the running of the sweeper thread in the Monitor and how long it takes the sweeper thread to run.

Suppose t_0 is the time at which a heartbeat is generated by a sender, and suppose t_f is the time at which the node fails.

$$t_f \in (t_0, \inf)$$

Suppose that t_a is the time at which the last heartbeat arrives at the monitor and SM is the amount of time it takes for the heartbeat to be transported over the network from the sender to the monitor.

$$t_a = t_0 + SM$$

Let t_d be the time at which this last heartbeat arrival is observed by the Monitor detection thread and used to generate a time-stamp that is used to test if the monitor times out. If the sweeper thread is not running when the heartbeat arrives, then the detection thread can be started immediately, and:

$$t_d = t_a$$

However, if the sweeper thread is running when the heartbeat arrives, then the detection thread cannot be started until the sweeper thread finished operation. Let SL represent the amount of time it takes to sweeper thread to run. In this case:

$$t_d \in (t_a, t_a + SL)$$

After the monitor heartbeat time-stamp has been generated, the sweeper thread could be started before this time-stamp value is placed in memory. Let t_p be the time at which this heartbeat is processed by the Monitor detection thread and placed in local memory. If sweeper thread preempts the placement of the heartbeat time stamp in memory by the detection thread, then

$$t_p \in (t_d, t_d + SL)\,.$$

If the sweeper thread does not preempt the placement of the heartbeat time stamp in memory by the detection thread,

$$t_p = t_d.$$

Let Th be the timeout threshold, and let t_t be the earliest time when the node will timeout if the sweeper thread runs.

$$t_t = t_p + Th$$

Let SI be the period of sweeper thread scheduling. Let t_{ss} be the first time at which the sweeper thread runs after t_t, the time when the node timed out. Because the sweeper thread is run at the highest priority,

$$t_{ss} \in (t_t, t_t + SI)\,.$$

Let t_{sf} be the time at which the sweeper stops running after the node times out.

$$t_{sf} = t_{ss} + SL$$

Let MN be the amount of time is takes for the node failure signal to be transported over the network from the Monitor tot he NSR. Let t_{NSR} be the time at which the failure signal is received at the NSR.

$$t_{NSR} = t_{sf} + MN$$

From these equations we can derive an expression for the maximum value of $t_{NSR} - t_f$, the maximum time for information about node failures to be sent to an NSR.

$$max(t_{NSR} - t_f) = SM + MN + Th + SI + 3SL$$

If there is a maximum delay associated with node failure notification, the NSR will be alerted to node failures within this maximum time (i.e., $max(t_{NSR} - t_f) < MaxDelay$) using an implementation of the NFD subsystem if:

$$SM + MN + Th + SI + 3SL < MaxDelay.$$

For our NFD implementation, our maximum delay objective was 100 msec., with each of these factors individually contributing to that total