Efficient and fault-tolerant distributed host monitoring using system-level diagnosis*

M. Bearden and R. Bianchini, Jr.
Department of Electrical and Computer Engineering,
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 USA
Telephone: 412-268-7105, Fax: 412-268-3890
mbearden@ece.cmu.edu, rpb@ece.cmu.edu

Abstract

This paper presents an efficient and fault-tolerant distributed approach to monitoring the status of processors in a network. The Distributed System Monitor (DSMon) is a distributed, decentralized program that gathers processor information, such as CPU load, user information, and network and disk statistics, in parallel at each processor and reliably distributes the information on-line to all fault-free processors. Information is filtered at each processor and distributed at different priorities to conserve communication resources. Fault-tolerance is achieved by applying the results of previous system-level diagnosis research. An on-line distributed system-level diagnosis algorithm that assumes the PMC fault model and a fully connected network is extended to consistently maintain user-defined state information in an unreliable environment.

DSMon has been implemented and currently operates on approximately 200 networked workstations in the Electrical and Computer Engineering Department at Carnegie Mellon University. The key results of this paper include the extension of a distributed system-level diagnosis algorithm for reliable broadcast of current global state, and the specification of the DSMon. A relaxed form of reliable broadcast, called condensed reliable broadcast, is introduced for guaranteeing delivery of the most recently broadcast update, without guaranteeing a complete history of all broadcast updates. The DSMon implementation is described, and its operation in an actual distributed network environment is analyzed. Extensions to this work include other fault and system models and applicability to other distributed applications requiring consistent distributed global state.

Keywords

Fault-tolerance, system-level diagnosis, distributed monitoring, reliable broadcast

*This research was supported in part by the National Science Foundation (NSF) under Grant CCR-9257973 and by a NSF Graduate Research Fellowship.

1 INTRODUCTION

This paper describes a fault-tolerant and fully distributed approach to monitoring the status of processors in a network. Typical network monitoring systems employ a centralized program that probes processors to verify their status. Lehman et al. (1992) alleviated the resulting bottleneck by distributing the monitoring task over a small set of processors. This work carries the idea further and proposes a fully decentralized approach. Information is gathered and filtered at each processor where it is generated, and the processors cooperate as peers to distribute changed information among themselves. Advantages of this decentralized approach include increased concurrency, the ability to tolerate processor failures, reduced network utilization, and greater scalability.

As the monitoring task is distributed across multiple processors, the probability of incorrect operation due to a single processor failure increases, unless a fault-tolerance scheme is utilized. System-level diagnosis research addresses the identification of faulty units in a system of interconnected components capable of testing one another. Distributed system-level diagnosis algorithms (Preparata et al., 1967; Dahbura, 1988; Bianchini and Buskens, 1992) provide for interconnected processors to locate faulty components and distribute fault information concerning component status to all fault-free processors. Recent research by Bianchini and Buskens (1992) has demonstrated that it is practical to execute a diagnosis algorithm on-line, concurrent with normal system operation, in a network of Unix workstations.

A program called the Distributed System Monitor (DSMon) is presented that utilizes a system-level diagnosis algorithm to reliably monitor the status of multiple unreliable host processors. DSMon distributes status information that is generated at each processor in a network, so that consistent information is available on-line from any fault-free processor in the network. A system-level diagnosis algorithm given by Bianchini and Buskens (1992) is extended to distribute arbitrary system state along with its diagnostic messages. It is shown that the modified algorithm, called Adaptive_DSD+, and a number of practical enhancements, meet the conditions for reliable broadcast.

Condensed updating and priority updating are two additional features of DSMon that reduce the number of network messages generated by the monitoring task. Condensed updating allows a message that represents a change in a monitored value to be discarded if it is distributed concurrently with a message indicating a more recent value. Priority updating allows the DSMon to distinguish between higher and lower priority updates and make appropriate time-versus-message cost trade-offs during update distribution. In particular, lower priority updates can be distributed with a longer time bound but using fewer messages, in order to conserve system resources. Message distribution can be configured so that the priority of each update message depends on changes that occur in the values being monitored.

The remainder of this paper is organized as follows. Section 2 presents a generalized model for a distributed system and formally defines a *condensed reliable broadcast* used for condensed updating. Section 3 provides an overview of the Adaptive DSD distributed system-level diagnosis algorithm presented in (Bianchini and Buskens, 1992), extends the algorithm to distribute arbitrary global state, and proves that it implements condensed reliable broadcast. In Section 4, the design and implementation of the Distributed System Monitor software are described. Section 5 gives experimental results. Conclusions and comments about future work are given in Section 6.

2 DISTRIBUTED SYSTEM MODEL

A distributed system is assumed to have the following three primary characteristics (Schroeder, 1993): multiple processors that fail independently, an interconnection network that allows the processors to communicate, and shared state that the processors cooperate to maintain. It is further assumed that processors and interprocessor communication are at least loosely synchronized; in other words, the time to execute a processing step or to deliver a message has a known bound, although processors may run at different speeds. It is assumed that only point-to-point interprocessor communication is provided by the network and that the network is logically fully connected. It is assumed that processors fail in a detectable way.

Examples of shared state that could be maintained by a distributed system include a list of CPU load values for dynamic load balancing applications (Ramamritham, et al., 1989) or a list of resources for a wide area network directory service (Obraczka et al., 1993). The goal of this work is to maintain a consistent copy of the system's global state, S^G, at each fault-free processor despite possible component failures. To meet this goal, an update of S^G must be reliably distributed from its correct originating processor to all correct processors, for any combination of correct processors and failure (repair) events.

The system's global state, $S^G = \{S_1^L, \ldots, S_i^L, \ldots, S_N^L\}$, is composed of local state information, S_i^L, that is sourced at each processor, P_i. S_i^L is a collection of state variables, V_{ij}, referring to the *j*th variable of S_i^L. Processor P_i is only permitted to modify the values of the variables in S_i^L, but maintains a copy of all global state variables and may be required to forward any variable to other processors. Communication of state variables is assumed to require multiple messages, such that several messages for an update to V_{ij} can be simultaneously transmitted between processors. We introduce the idea of a "condensed" reliable broadcast mechanism for distributing an update to variable V_{ij}:

Definition: A *condensed reliable broadcast* of shared state information requires that four properties be satisfied for a message sent by processor P_i at time t containing a value for variable V_{ij}, denoted message V_{ij}^t:

- CRB1. *Integrity*: A message is accepted by a fault-free processor only if the message was sent by a fault-free processor;
- CRB2. *Condensed Partial Ordering*: A message V_{ij}^x is accepted by a fault-free processor only if a message V_{ij}^y, $y > x$, has not been previously accepted;
- CRB3. *Validity:* If message V_{ij}^x is sent by a fault-free processor, then V_{ij}^x will eventually be accepted by (at least) one other fault-free processor, unless some message V_{ij}^y, $y > x$, is eventually received at each fault-free processor.
- CRB4. *Agreement*: If message V_{ij}^x is accepted by a fault-free processor, then either message V_{ij}^x or some message V_{ij}^y, $y > x$, will eventually be accepted by all fault-free processors, including any processor that is repaired at time r, $r > x$.

The condensed reliable broadcast[1] is less restrictive in its operation than a "complete" reliable broadcast, one that eventually delivers the same set of messages to each processor (Hadzilacos and Toueg, 1993). Properties CRB3 and CRB4 guarantee delivery of the most recently generated value for a particular variable but allow that some processors may never receive values whose distribution overlaps in time with the distribution of more recently generated values. For example, if a fault-free processor generates three update messages, *M1, M2,* and *M3*, for a

[1] To the authors' knowledge this is the first formal definition of condensed broadcast. The idea was informally mentioned by Cheriton and Skeen (1993).

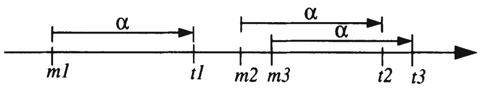

Figure 1 Condensed reliable broadcast example.

single variable at the times marked (respectively) *m1*, *m2*, and *m3* on the timeline in Figure 1, then all other fault-free processors must receive *M1* and *M3*, but some or all processors may not receive *M2*. Each period labeled α is the assumed maximum bound for the time to deliver a broadcast message. Because a synchronous distributed system has been assumed, by time *t1* all processors must have received *M1*. By time *t2*, all processors must have received either *M2* or *M3*, and all must have received *M3* by time *t3*. Condensed reliable broadcast is sufficient for a monitoring environment that maintains the current value of s^G, rather than a history of changes to s^G, and can result in significantly lower overhead than complete broadcasting when changes to a state variable are broadcasted frequently.

3 DISTRIBUTED SYSTEM-LEVEL DIAGNOSIS

System-level diagnosis research focuses on the identification of faulty components in a system of interconnected nodes, or processors, that are capable of testing one another. A fault state $s_i \in$ {faulty, fault-free} is assigned to each of the system components. Directed edges, or arcs, of a *testing assignment* represent an inter-node test that is performed by one node on another node. Tests results are mapped to node fault states by a *fault model* that dictates possible test outcomes based on the fault states of the nodes (and edges) involved in a test. Seminal work on the system-level diagnosis problem appeared in (Preparata et al., 1967) and (Hakimi and Amin, 1974). Surveys can be found in (Friedman and Simoncini, 1980; Dahbura, 1988; Barborak et al., 1991).

In *distributed* system-level diagnosis, each node in a distributed system independently diagnoses all other nodes using locally maintained test result information. Each node directly tests a subset of the system nodes and utilizes test results reported by other nodes. The distributed system-level diagnosis algorithms presented in (Hosseini et al., 1984; Bianchini and Buskens, 1992; Bagchi, 1992; Stahl et al., 1992) utilize a common strategy for distributing test result reports. The algorithms require that nodes validate diagnosis information received from nodes that they test. Diagnosis information flows "backward" along arcs in the testing assignment. The algorithms guarantee that a strongly connected testing assignment is established among the fault-free nodes, permitting each node to receive valid diagnosis information from all other fault-free nodes. Every fault-free node forms a *syndrome*, or set of test results, without directly testing every other node. The distributed diagnosis algorithms guarantee that a consistent syndrome exists at every fault-free node. This property of distributed diagnosis algorithms is exploited in this paper to implement the condensed reliable broadcast.

An overview of the *Adaptive DSD* algorithm (Bianchini and Buskens, 1992) to be used as the foundation of this work is given in Section 3.1. Section 3.3 describes modifications to the algorithm required to maintain the consistency of arbitrary information in addition to the diagnosis information.

3.1 The Adaptive DSD algorithm

Using the Adaptive DSD algorithm, the nodes in a distributed system collaborate to produce a testing assignment that adapts to the current fault situation by "testing around" faulty nodes. Tests are performed periodically and assume the PMC fault model (Preparata et al., 1967): tests performed by fault-free nodes are always correct, and tests performed by faulty nodes produce arbitrary results. For on-line operation, it is further assumed that a node cannot fail and recover in an undetected fashion between consecutive tests. The system nodes are organized logically in a ring topology. Each node tests consecutive nodes in the ring until a test result returns fault-free. The resulting testing assignment contains a directed cycle that includes all fault-free nodes and is minimum in the number of tests required. Figure 2 shows the testing assignment that is achieved in a network of eight nodes, three of which are faulty.

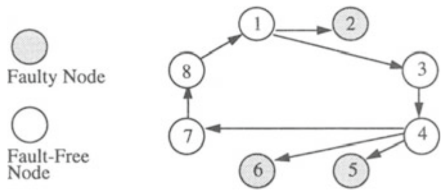

Figure 2 Adaptive DSD example testing assignment.

The Adaptive DSD algorithm, executing at N nodes, correctly diagnoses up to N-1 faulty nodes and issues N tests, the minimum number required for diagnosis, since each node is tested once. Syndromes information is forwarded between processors during the period between two consecutive tests, in the reverse direction of the tests. The syndromes information consists of the N-element Tested_Up array. Total message count is N^2, since N Tested_Up elements are transmitted to each of the N nodes. The diagnosis latency of the algorithm is the time required for all fault free nodes to achieve correct diagnosis after a fault event is detected. Correct diagnosis is achieved when the Tested_Up array is consistent at all fault-free nodes, after an updated element has been forwarded via the testing assignment to all fault free nodes. Due to the cyclic testing assignment, the diagnosis latency is bounded by NT_P, for an interval of T_P between periodic tests. Detailed performance bounds are given by Bianchini and Buskens (1992).

3.2 Algorithm enhancements

Although the Adaptive DSD algorithm is optimal in terms of test count, enhancements can be made to improve message count and diagnosis/update latency. The following enhancements are made as suggested in (Bianchini et al., 1990) and (Bianchini and Buskens, 1992).

Event-driven information updating

It is expected that elements of the syndrome (Tested_Up array) will typically remain constant over several testing periods. In the base algorithm, the same value is transmitted multiple times for each entry that remains constant. Message count is reduced by forwarding only those entries with information that has changed. It is demonstrated in (Bianchini et al., 1990) that information updating operates correctly if a node only forwards information under two circumstances: (1) An array element is forwarded if its value has changed, and (2) all array elements are forwarded to a newly recovered node that may have lost its previous state. Forwarding only changed information reduces the total message count from N^2 messages per test round to $N\Delta v$, where Δv is the total number of Tested_Up entries that change.

Diagnosis latency is reduced by immediately transmitting any changed diagnosis information that is generated or forwarded. In the base algorithm, new information is forwarded by a node only after the next periodic test is issued. Using event-driven updates, information is forwarded immediately after its value changes. For proper operation using the PMC fault model, the update message must be validated by arriving between two consecutive fault-free tests (Hosseini et al., 1984). To implement event-driven operation, the next periodic test of a node that sends an update is scheduled earlier to occur immediately after the update is received. Event-driven updating reduces diagnosis latency from NT_p to NT_{fwd}, where T_{fwd} is the time required to validate and forward a message at each node.

Tree-based redundant update forwarding

Update and diagnosis latency may be reduced by sending information along redundant communication paths that are shorter in length than the testing assignment forwarding path (ring). An asymmetric forwarding topology is used that represents a K-ary tree of the fault-free processors. Asymmetrically forwarded messages are sent in addition to the messages forwarded around the testing assignment ring, reducing the longest forwarding path from N to $\lceil \log_K N \rceil$ nodes. Each extra message transmission potentially requires the scheduling of extra tests to insure that each forwarded message is validated by arriving between fault-free tests. The asymmetric forwarding topology is adapted whenever a node changes state. Asymmetric forwarding reduces the diagnosis latency to $\lceil \log_K N \rceil T_{fwd}$ and requires an additional $[(K-1)/K]N$ messages and up to twice that many additional tests.

Time stamped updates

The enhancements described above yield significant improvements in run-time execution but can allow Tested_Up updates to arrive out of order. Due to asymmetric forwarding, an old message can arrive at a processor on a longer path than a newer message and overwrite the newer value, even if FIFO point-to-point message delivery is assumed. To alleviate this problem, a logical clock, h_i, with strictly increasing time values, is maintained at each processor P_i (Hakimi and Nakajima, 1984; Lamport, 1978). Every time that a processor forwards a Tested_Up entry, h_i is incremented and sent with the entry. Another processor that receives the array entry uses the accompanying logical clock value to determine its age, relative to any previous value that was accepted for that entry, and accepts only more recent entries.

For correct operation, a processor that has restarted after failing must be able to determine the highest value of h_i that it used before failing and use a higher value for all subsequent messages. Initially, the processor resets its logical clock to $h_i = 0$ and forwards any information using low logical clock values. Using the event-driven forwarding enhancement, the newly restarted processor must eventually receive all Tested_Up values from another fault-free node. If the node receives a higher logical clock value for its entry, then its current logical clock is updated to the new value plus one, and its current information is forwarded again.

3.3 Incorporating global state: Adaptive DSD+

A minor modification is made to the Adaptive DSD algorithm to reliably distribute global state information with diagnosis information. The modified Adaptive DSD+ algorithm is given in Figure 3. The unhighlighted portion represents the original algorithm (Bianchini and Buskens, 1992). The S_Table data structure is added and contains the global system state S^G. Entry S_Table[i] contains the local state S_i^L for processor P_i and is modified independently of Adaptive DSD+ execution. Steps 2.3 and 4.1.2 are added to distribute S_Table in the same way that the Tested_Up array is distributed. The algorithm operates at each node by periodically testing for the next fault-free node and receiving its Tested_Up and S_Table arrays. Each of the performance enhancements described in the preceeding section are applied to the S_Table entries as well as to the Tested_Up entries. Thus, changes to S_Table are forwarded to all fault-free nodes within the diagnosis latency bound of Adaptive DSD.

```
/* Adaptive DSD+                                        */
/* Executed at each $n_x$, $0 \le x \le N-1$            */
/* at predefined testing intervals.                     */
1.      $y = x$;
2.      repeat {
2.1.        $y = (y+1) \bmod N$;
2.2.        request $n_y$ to forward Tested_Up$_y$ to $n_x$;
2.3.        request $n_y$ to forward S_Table$_y$ to $n_x$;
2.4.        } until ($n_x$ tests $n_y$ as fault-free);
3.      Tested_Up[$x$] = $y$;
4.      for $i = 0$ to $N-1$
4.1.        if ($i \neq x$)
4.1.1.          Tested_Up[$i$] = Tested_Up$_y$[$i$];
4.1.2.          S_Table[$i$] = S_Table$_y$[$i$];
```

Figure 3 The Adaptive DSD+ algorithm. Additions to Adaptive DSD are highlighted.

3.4 Algorithm correctness

The Adaptive DSD algorithm is proven correct in (Bianchini and Buskens, 1992), since the algorithm maintains consistent syndrome information (the Tested_Up array) at all fault-free nodes. The proof that Adaptive DSD+ must maintain consistent state information (Tested_Up and S_Table arrays) at all fault-free nodes is a simple generalization of the Adaptive DSD proof, since Adaptive DSD+ sends all of the state information with the syndrome information. In this section it is proven that the Adaptive DSD+ algorithm with the enhancements given in Section 3.2 satisfies the requirements for condensed reliable broadcast, given in Section 2, of the Tested_Up and S_Table entries. The PMC fault model with periodic testing (Hosseini et al., 1984) is assumed, such that a node cannot fail and recover in an undetected fashion in the interval between two tests.

Theorem 1: The enhanced Adaptive DSD+ algorithm satisfies CRB1 (Integrity) for all entries of S_Table.

Proof: A message is accepted by processor P_a from processor P_b either if P_a is periodically testing P_b or P_b asymmetrically forwards the message to P_a. If P_a is periodically testing P_b, then the message must be valid due to the fault model assumption. If P_b asymmetrically forwards the message to P_a, then the algorithm requires P_a to test P_b before and after the message is transmitted and thus the message must be valid due to the fault model assumption.

Theorem 2: The enhanced Adaptive DSD+ algorithm satisfies CRB2 (Condensed Partial Ordering) for all entries of S_Table.

Proof: Theorem 2 is a direct consequence of the time stamping enhancement. Consider two values V_{ij}^{x} and V_{ij}^{y} generated for V_{ij} with $x < y$, i.e. V_{ij}^{x} is generated first. Assume that both values arrive at processor P_a. If P_a receives and accepts V_{ij}^{y} first, then receives V_{ij}^{x}, V_{ij}^{x} will be ignored, since by the time stamping protocol, it must have a lower logical clock value.

Theorem 3: The enhanced Adaptive DSD+ algorithm satisfies CRB3 (Validity) for all entries of S_Table.

Proof: Consider a processor P_i that generates V_{ij}^{y} and does not generate V_{ij}^{y}, $y > x$. According to the Adaptive DSD+ algorithm, the previous fault-free processor, P_h, in the testing assignment must periodically test P_i. Assume that P_i remains fault-free during the period from the periodic test just prior to and just after V_{ij}^x is generated. Then by forwarding condition (1) of the event-driven information updating enhancement P_h must receive and accept V_{ij}^x. If P_h recovers from a failure after V_{ij}^x is generated, then by forwarding condition (2) of the information updating enhancement, it still must receive V_{ij}^x, since all entries of S_Table including V_{ij}^x, are forwarded to P_a.

Theorem 4: The enhanced Adaptive DSD+ algorithm satisfies CRB4 (Agreement) for all entries of S_Table.

Proof: It is proven in Theorem 3, that a message, V_{ij}^x, generated by processor P_b will eventually be received by the previous fault-free processor, P_a, in the testing assignment, even if node P_a recovers after V_{ij}^x is generated. If P_a becomes faulty immediately after receiving V_{ij}^x, the testing assignment is eventually adjusted so the node previous to P_a tests P_b and receives V_{ij}^x, by forwarding condition (2) of the information updating enhancement. Between processor failures and repairs, the testing assignment of the Adaptive DSD+ algorithm forms an ordering of all of the fault-free nodes. Theorem 3 is applied to each previous node in the testing assignment to prove that all fault-free nodes must receive V_{ij}^x, even in the presence of fault events.

By Theorems 1 through 4, the Adaptive DSD+ algorithm satisfies the condensed reliable broadcast protocol for all elements of the S_Table array. ■

4 OVERVIEW OF THE DISTRIBUTED SYSTEM MONITOR

The Distributed System Monitor, DSMon, is a fully distributed program that allows a user at any fault-free host processor in a network to obtain state information about any host processor in the system. Consider the model of a uniprocessor monitoring system given in Figure 4(a). The system consists of one or more *Monitor* modules that generate information about the state of the host processor and one or more *Query* modules that log, analyze, or present the host information for a user. DSMon is implemented in a distributed system by the addition of one *Distribution* module, as shown in Figure 4(b), at every host processor. Each Distribution module receives local information generated by Monitor modules and forwards that information to any local Query modules and to other Distribution modules located on remote processors. The Distribution modules reliably distribute the state information among processors by utilizing the condensed reliable broadcast protocol implicitly provided by continuous execution of the Adaptive DSD+ algorithm (see Section 3). Thus, the Distribution modules perform inter-processor testing and forward state variables and diagnosis information. State variables are guaranteed to always

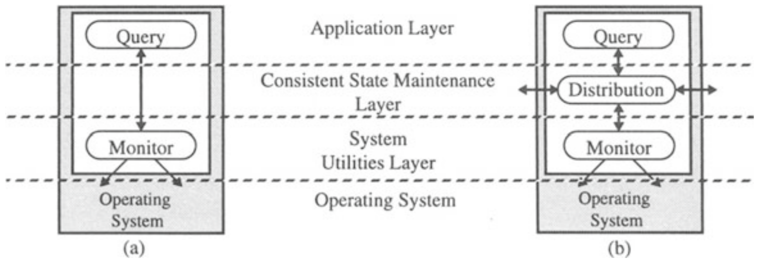

Figure 4 DSMon program modules.

be current, unless the diagnosis information identifies a node as faulty, in which case the values for that processor are identified as invalid. The resulting view of system state available to each Query module is the complete set of current values generated for all hosts. DSMon guarantees that the state views at any two fault-free hosts are consistent after each state change is distributed.

4.1 Filters and distribution priorities

To conserve network resources, the DSMon program supports variable filtering (Mansouri-Samani and Sloman, 1993) to minimize the amount of monitoring data that is distributed. Programmable filters similar to those specified in (ISO/IED, 1990) are part of the DSMon configuration for every host. The filters allow the Distribution modules of the DSMon to determine online which values generated by Monitor modules should actually be distributed. This can be used to eliminate monitor information that is too detailed to be of interest to users, such as slight variations that occur frequently in a monitored value. A complete description of DSMon monitor variable filtering is given in (Bearden, 1993), including specifications of the filter functions supported.

To further reduce network resource utilization, DSMon supports information distribution at two priority levels: EVENT and TRICKLE, described in Table 1. Updates given EVENT priority are distributed using the event-driven tree-based forwarding mechanism given in Section 3.2. EVENT distribution is appropriate for distributing higher priority changes in variables and TRICKLE distribution is more appropriate for less important updates. Diagnosis information is always forwarded at the EVENT priority level. The TRICKLE priority level is supported to utilize fewer communication resources and results in longer distribution times. Updates given TRICKLE priority are queued at each node and are only forwarded to another node when they can be "piggybacked" with a non-TRICKLE message that is transmitted to that node. The typical delay at each node is low since the periodic testing performed by the Adaptive DSD+ algorithm generates a test message and response once per testing period. TRICKLE values can also be forwarded with EVENT messages.

Table 1 DSMon variable distribution actions

Action	*Result*
TRICKLE (V_{ij})	Send new value for V_{ij} at some future time: either when the next periodic test occurs, or sooner if it can be piggybacked with an EVENT priority message.
EVENT(V_{ij})	Send new value for V_{ij} immediately.

4.2 Expected performance

The update latency of a variable is the time required for an update to be received by all nodes after its generation. Bounds on update latency for distribution actions, given in Table 2, are derived directly from the Adaptive DSD+ bounds presented in Section 3.1. EVENT priority latency is $\log_K NT_{\mathrm{fwd}}$, the same as diagnosis latency. In the worst case, TRICKLE priority variables are forwarded with test results, so TRICKLE priority latency is bounded by NT_{p}. Total message count requires N messages for each of the Δv variable events and $2N$ messages for test request and response.

Table 2 DSMon performance bounds

Message Count	$(1+\frac{K-1}{K})N\Delta v + 2N$
Event Update Latency	$\log_K N T_{\mathrm{fwd}}$
Trickle Update Latency	NT_{P}

5 DSMON IMPLEMENTATION AND EXPERIMENTATION

DSMon has been running in the Carnegie Mellon University (CMU) ECE department since May 1993 on approximately 200 workstations, including DEC 5000, DEC Alpha, and IBM PowerPC workstations, running variants of the Unix operating system. Recently a small number of personal computers running Linux and Microsoft Windows have also been integrated with the system. The Distribution module is approximately 5,000 lines of C source code and is implemented as low-priority background processes (via the Unix "nice" utility) that are started when each workstation is booted. The Monitor process is currently an SNMP (Case et al., 1989) agent that has been customized to obtain host processor information. The Query module is a program that is invoked at any workstation and allows the user to view the status of a subset of hosts selected via a graphical user interface.

The interfaces for interprocess and network communication use the Berkeley socket interface and presently support Ethernet IP/UDP protocols. Appropriate modifications to the communication source code modules will allow the program to run on any system that has a C compiler. A retry-with-acknowledgment scheme is implemented by each processor to make datagram (packet) transmission reliable. The configuration for each workstation executing DSMon is read from a file that is processed by each DSMon module. The configuration file contains a list of hosts running DSMon, operational parameters required for communication and diagnosis, and definitions of variables and filters for each workstation.

5.1 The CMU ECE monitor variable set

Presently, three monitor variables are monitored by DSMon for each workstation in the ECE department of CMU. The DSMon filters used for the variables are given in Table 3. Each variable is sampled once per minute by the Monitor process at each workstation. The filter column identifies the change in value required, Δ*variable*, to result in distribution of a new variable value. For example, the entry for "Users" shows that any change in the list of users logged in to a workstation will be detected by the local DSMon within one minute. Filtered values will be forwarded at EVENT priority to the other workstations executing DSMon. The filters given in

Table 3 The variables and filters used to monitor the ECE network at CMU.

Variable	*Description*	*Filter algorithm*

CPU	Processor load (average number of jobs in Unix run queue)	If (ΔCPU > .50) EVENT(CPU);
		If (CPU ≥ .50) ∧ (ΔCPU > .20) TRICKLE(CPU);
		If (CPU < .50) ∧ (ΔCPU > .10) TRICKLE(CPU);

Disk	% of local disk free	If (ΔDisk > .50) EVENT(Disk);
		If (Disk ≥ 92) ∧ (ΔDisk ≥ 3) TRICKLE(Disk);
		If (Disk < 92) ∧ (ΔDisk ≥ 5) TRICKLE(Disk)

Users	List of users logged in	If (ΔUsers) EVENT(Users);

Table 3 have resulted in a average DSMon load of approximately five "User" EVENTs, three "CPU" EVENTs, and one "Disk" EVENT per minute per 100 ECE workstations during peak work hours.

5.2 Experimentation

To illustrate the run-time behavior of DSMon a number of experiments were run in the CMU ECE network. Experimentation focused on communication overhead, in terms of average packet count and bytes sent per testing period, and diagnosis and variable distribution latency, measured in seconds. The experiment figures graph communication overhead (packets and bytes sent) ***per workstation*** as a function of elapsed time for each of three experiments run on 100 workstations. In all experiments, inter-node tests are performed every 30 seconds. EVENT priority updates are forwarded using a binary tree asymmetric forwarding path. For each workstation the number of packets, total number of bytes contained in the packets, and detected fault events and state changes were recorded at 10 second intervals.

Experiment 1, given in Figure 5, illustrates the network load resulting from workstation failures and recoveries alone. During periods without fault events, the network load is two packets per node per test period, corresponding to a test request and a test response message. A single workstation was halted and restarted at 70 and 158 seconds respectively. The graph indicates the times at which the failure and recovery were detected and times at which all fault-free workstations had updated their diagnosis. The load resulting from multiple fault events is shown following three simultaneous failures and two simultaneous recoveries.

Experiment 2, given in Figure 6, illustrates the network load that results from distributing updates of monitor variables. The grey bar at the bottom of the graph indicates when new values are generated for state variables. Times that are labeled "Reinformed" indicate when all workstations have reached the same consistent view of the global state. Each EVENT message causes every processor to receive and forward one update packet and to execute a test to validate the message. A single EVENT message causes each node to send approximately two additional packets in a test period.

During the period labeled "A" in Experiment 2, global consistency is maintained and all processors have precisely the same values for all monitor variables. During the period labeled "B," a number of TRICKLE values are generated and are initially forwarded. The average number of

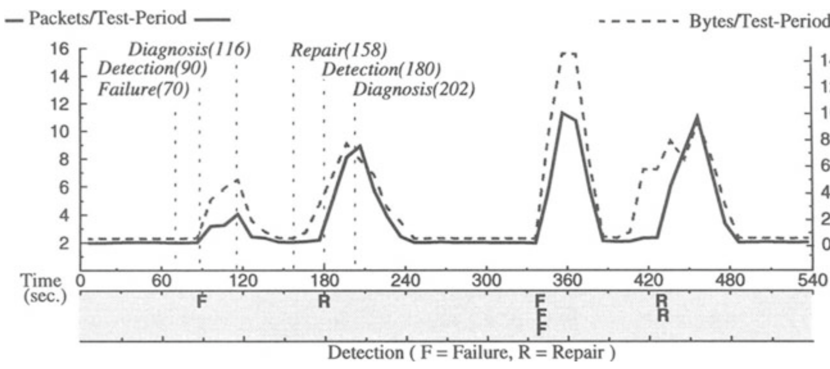

Figure 5 Experiment 1: network load from fault events.

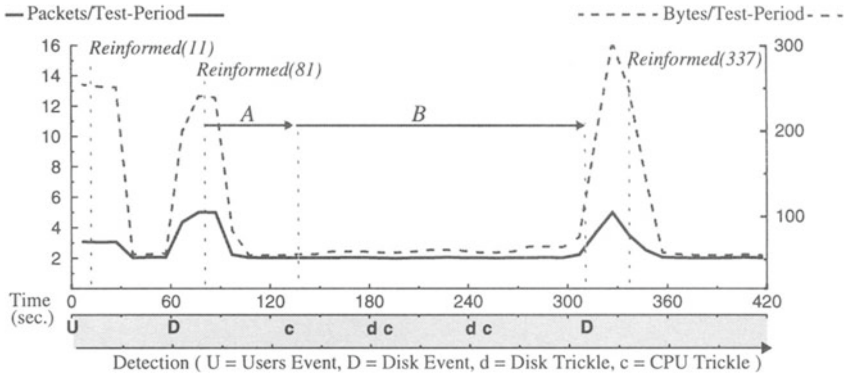

Figure 6 Experiment 2: network load due to variable updates.

bytes per packet increases slightly, but packet count remains the same due to the combining of TRICKLE messages with test response messages. During this period, some processors have slightly different values for "CPU" and "Disk" variables, but all values are guaranteed to be within the EVENT updating range specified for each variable.

Figures 7 illustrates the network load during a ten minute interval of normal DSMon and network operation for the three variables given in Table 3. The generation times for fault, EVENT, and TRICKLE updates are indicated in the grey bar at the bottom of the figure.

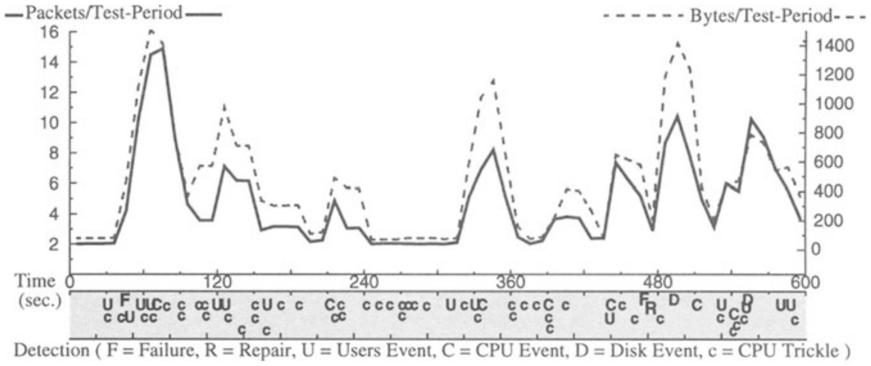

Figure 7 Experiment 3: network load during normal operation.

6 CONCLUSIONS

This paper presents an application of distributed system-level diagnosis theory to the problem of creating fault-tolerant distributed monitoring software. The DSMon is a distributed program that reliably communicates current values monitored in parallel at multiple host processors to all fault-free computers in a system. A specialized light weight form of reliable broadcasting, called condensed reliable broadcasting, is introduced to conserve network resources. Theoretical and experimental validation of the system-level diagnosis based approach are given. Operational overhead is low and has been shown through practice to be suitable for use in a general purpose network of workstations and personal computers.

DSMon is currently restricted to operation in a logically fully connected network because of the distributed diagnosis algorithm, Adaptive DSD, chosen as the basis for this work. Future implementations of DSMon will integrate distributed diagnosis algorithms (Stahl et al., 1992; Buskens and Bianchini, 1993) that place less restriction on network connectivity and that have more practical fault model assumptions. Other planned experimentation includes the addition of update coordination for state variables that are owned by more than one processor, and for on-line reconfiguration. The Adaptive DSD+ algorithm presented in this paper, and other extended distributed diagnosis algorithms, are expected to prove useful as a tool for implementing a variety of distributed applications that require consistent global state.

7 REFERENCES

Bagchi, A. (1992) A distributed algorithm for system-level diagnosis in hypercubes. *Proceedings of the 1992 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, July, 1992, Amherst, Massachussetts, 106-113.

Barborak, M., Malek, M., and Dahbura, A. (1991) The consensus problem in fault-tolerant computing. *ACM Computing Surveys* (USA), **25(2)** (June), 171-220.

Bearden, M. (1993) The Distributed System Monitor: a practical implementation of system-level diagnosis, Research Report No. CMUCSC-93-7, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Bianchini, R., Jr., Goodwin, K., and Nydick, D. (1990) Practical application and implementation of distributed system-level diagnosis theory. *Proceedings, 20th International Symposium on Fault-Tolerant Computing*, IEEE, Boston, June, 332-9.

Bianchini, R. and Buskens, R. (1992) Implementation of on-line distributed system-level diagnosis theory. *IEEE Transactions on Computers,* **41(5)** (May), 616-26.

Buskens, R. and Bianchini, R., Jr. (1993) Distributed on-line diagnosis in the presence of arbitrary faults. *Proc., 23rd Int. Symp. on Fault-Tolerant Computing*, IEEE, June, 470-9.

Case, J., Davin, C., Fedor, M., and Schoffstall, M. (1989) Internet network management using the simple network management protocol. *Proceedings, 14th IEEE Conference on Local Computer Networks*, Mineapolis, Minnesota (USA), October, 156-9.

Cheriton, D. and Skeen, D. (1993) Understanding the limitations of causally and totally ordered communication. *Proceedings of the 14th Symp. on Operating Systems Principles*, ACM, December, in *Operating Systems Review,* **27(5)**, 44-57.

Dahbura, A. (1988) System-level diagnosis: a perspective for the third decade, in *Concurrent Computations: Algorithms, Architecture, and Technology*, (S. Tewksbury, et al., eds.), Plenum Press.

Friedman, A. and Simoncini, L. (1980) System-level fault diagnosis. *IEEE Computer*, March 1980, 47-53.

Hadzilacos, V. and Toueg, S. (1993) Fault-tolerant broadcasts and related problems, in *Distributed Systems,* 2.0 edition, (ed. S. Mullender), ACM Press, New York.

Hakimi, S. and Amin, A. (1974) Characterization of connection assignment of diagnosable systems. *IEEE Transactions on Computers,* **C-23(1)** (Jan.), 86-88.

Hakimi, S. and Nakajima, K. (1984) On adaptive system diagnosis. *IEEE Transactions on Computers,* **C-33(3)** (March), 234-40.

Hosseini, S., Kuhl, J., and Reddy, S. (1984) A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *IEEE Trans. on Comp.*, **C-33(3)** (Mar.), 223-33.

ISO/IED (1990) Information Technology - Open Systems Interconnection - Systems Management Part 5: Event Report Management Function. ISO/IED DIS 10164-5, October 1990.

Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* (USA), **21(7)** (July), 558-65.

Lehman, R., Carpenter, G., and Hien, N. (1992) Concurrent network management with a distributed management tool. *Proceedings of the 6th Systems Administration Conference (LISA VI)*, USENIX Assocation, October, Long Beach, California, 235-44.

Mansouri-Samani, M. and Sloman, M. (1993) Monitoring distributed systems. *IEEE Network*, November 1993, 20-30.

Obraczka, K., Danzig, P., and Li, S. (1993) Internet resource discovery services. *IEEE Computer* (USA), **26(9)** (Sept.), 8-22.

Preparata, F., Metze, G., and Chien., R. (1967) On the connection assignment problem of diagnosable systems. *IEEE Trans. on Electronic Computers*, **EC-16(6)**, (Dec.), 848-54.

Ramamritham, K., Stankovic, J., and Zhao, W. (1989) Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. on Computers*, August 1989, 1110-1123.

Schroeder, M. (1993) A state-of-the-art distributed system: computing with BOB, in *Distributed Systems,* 2.0 edition, (ed. S. Mullender), ACM Press, New York.

Stahl, M., Buskens, R., Bianchini, R. (1992) On-line diagnosis in general topology networks. *Proceedings of the 1992 IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, July, 1992, Amherst, Massachussetts, 114-121.

8 BIOGRAPHY

Mark Bearden is currently a Ph.D. candidate in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received his B.S.E.E. degree from The University of Alabama (Tuscaloosa) in 1991 and his M.S.E.C.E. at Carnegie Mellon University in 1993. During 1994 he interned in the Distributed Software Research Department at AT&T Bell Laboratories in Murray Hill, New Jersey. Mark is a student member of the IEEE and an alumnus of The University of Alabama Computer-Based Honors Program. His research interests include developing efficient decentralized algorithms for serverless networks.

Ronald P. Bianchini, Jr. is currently an Associate Professor in the Electrical and Computer Engineering Department at Carnegie Mellon University. He received his B.S. degree in Electrical Engineering from the Massachusetts Institute of Technology in 1983 and his M.S. and Ph.D. degrees in Electrical and Computer Engineering from Carnegie Mellon University in 1985 and 1989, respectively. His research interests include distributed fault tolerant computing, distributed system-level diagnosis, computer networks and advanced telecommunication switching architectures. He has consulted for AT&T Bell Laboratories during the summers of 1990 and 1991 in the area of distributed fault diagnosis. Dr. Bianchini is a member of the IEEE Computer Society, the Sigma Xi Honor Society and was nominated to the Eta Kappa Nu Honor Society in 1983.