Automated Rule-Based Diagnosis through a Distributed Monitor System

Gunjan Khanna, Mike Yu Cheng, Padma Varadharajan, Saurabh Bagchi, Miguel P. Correia, and Paulo J. Veríssimo, *Senior Member, IEEE*

Abstract—In today's world, where distributed systems form many of our critical infrastructures, dependability outages are becoming increasingly common. In many situations, it is necessary to not only detect a failure but also to diagnose the failure, that is, to identify the source of the failure. Diagnosis is challenging, since high-throughput applications with frequent interactions between the different components allow fast error propagation. It is desirable to consider applications as blackboxes for the diagnostic process. In this paper, we propose a Monitor architecture for diagnosing failures in large-scale network protocols. The Monitor only observes the message exchanges between the protocol entities (PEs) remotely and does not access the Internal Protocol state. At runtime, it builds a causal graph between the PEs based on their communication and uses this together with a rule base of allowed state-transition paths to diagnose the failure. The tests used for the diagnosis are based on the rule base and are assumed to have imperfect coverage. The hierarchical Monitor framework allows distributed diagnosis handling failures at individual Monitors. The framework is implemented and applied to a reliable multicast protocol executing on our campuswide network. Fault injection experiments are carried out to evaluate the accuracy and latency of the diagnosis.

Index Terms—Distributed system diagnosis, runtime monitoring, hierarchical Monitor system, fault-injection-based evaluation.

1 Introduction

The wide deployment of high-speed computer networks has made distributed systems a fundamental infrastructure in today's connected world. The infrastructure, however, is increasingly facing the challenge of dependability outages resulting from both accidental and malicious failures. These two classes are collectively referred to as *failures* in this paper. The potential causes of accidental failures are hardware failures, software defects, and operator failures, including misconfigurations. The malicious failures may be due to external attackers or internal attackers. The financial consequences of failures to distributed infrastructure can be gauged from a survey by the Meta Group Inc. of 21 industrial sectors in October 2000 [1], which found that the mean loss of revenue due to an hour of computer system downtime is $1,010,000. Compare this to the average cost of $205 per hour of employee downtime. Also, compare the cost today to the average of $82,500 in 1993 [2], and the trend becomes clear.

In order to build a robust infrastructure capable of tolerating the two classes of failures, it is required that *detection* and *diagnosis* primitives be provided as part of a fault-tolerant infrastructure. Following the definitions in [24], a *fault* is an invalid state or a bug underlying in the system, which, when triggered, becomes an *error*. A *failure* is an external manifestation of an *error* manifested to the user. A *failure* in a distributed system may be manifested at an entity that is distant from the one that was originally in error. This is caused by *error* propagation between the different communicating entities. The role of the diagnosis system is to identify the entity that originated the failure. The *diagnosis* problem is significant in distributed applications that have many closely interacting PEs, since the close interactions facilitate error propagation.

In our target approach, we structure the combined system into two clearly segmented parts with well-defined mutual interactions—an observer or Monitor system, which provides *detection* and *diagnosis*, and an observed or payload system, which comprises the *protocol entities* (PEs)—that is, the processes that implement the functionality of the distributed system. This paper builds the *diagnosis* functionality to complement the *detection* functionality in the Monitor system that was presented earlier in [4]. The monitoring and collection of the system state in various forms have been widely studied (including [4], [5], [6], [41]), but this paper addresses the problem of providing diagnosis based on the system state.

There are several design motivations for the Monitor system. First, it is desirable that the Monitor system operates asynchronously to the payload system so that the system's throughput does not suffer due to the checking overhead. Second, there is a requirement of fast detection and diagnosis so that substantial damage due to cascaded failures is avoided. Third, the Monitor system should not be intrusive to the payload system. This rules out the possibility of making changes to the PEs or creating special tests that they respond to. Instead, this argues in favor of having the payload system be viewed as a blackbox by the

- *G. Khanna, M.Y. Cheng, P. Varadharajan, and S. Bagchi are with the School of Electrical and Computer Engineering, Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907. E-mail: {gkhanna, mikecheng, pvardha, sbagchi}@purdue.edu.*
- *M.P. Correia and P.J. Veríssimo are with the Departamento de Informática, Universidade de Lisboa, Campo Grande, 1749-016 Lisboa, Portugal. E-mail: {mpc, pjv}@di.fc.ul.pt.*

Manuscript received 17 Sept. 2006; revised 8 May 2007; accepted 23 May 2007; published online 22 June 2007.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0134-0906. Digital Object Identifier no. 10.1109/TDSC.2007.70211.

Monitor system. Although it is possible to build very optimized and specialized fault-tolerant mechanisms for specific applications (for example, like our earlier work in [3], with the Tree-based Reliable Multicast (TRAM) protocol), such solutions do not generalize well across applications. Thus, it is important to design the Monitor system to have an application-neutral architecture and with ease of deployment across applications.

In today's distributed systems, the machines on which the applications are hosted are heterogeneous in nature, the applications often run legacy code without the availability of their source, and the systems are of very large scales with soft real-time guarantees, making it particularly challenging to meet all of the above goals. In this paper, we propose a generic Monitor architecture to provide diagnosis primitives to distributed applications, meeting all the design requirements mentioned above.

We use a hierarchical Monitor architecture to perform a diagnosis of failures in the underlying protocol. The Monitor snoops on the communication between the PEs and performs diagnosis of the faulty PE once a failure is detected. We use the terminology "the Monitor *verifies* a PE" to mean that the Monitor provides the *detection* and the *diagnosis* functionalities to the PE. The Monitors treat the PEs as blackbox, and only the causal relation among the messages deduced from the send-receive ordering, along with a rule base containing correctness and quality-of-service (QoS) rules, are used to perform the diagnosis. For diagnosis, the PEs are not exercised with additional tests, since that would make the Monitor system more invasive to the application protocol. Instead, a state that has already been deduced by the Monitors during a normal operation through the observed external messages is used for the diagnostic process.

A low-level Monitor, called a *Local Monitor* (LM), directly verifies a PE, whereas a higher level Monitor, called an *Intermediate Monitor* (IM), matches rules that span multiple LMs. The Monitor architecture is generic and applicable to a large class of message passing-based distributed applications. It is the specification of the rule base that makes the Monitor specialized for an application. The Monitors coordinate to perform a distributed diagnosis if the verified PEs lie under the verification domains of different Monitors. We assume that failures may occur in the Monitor system as well and use replication to mask them. We enforce a hybrid failure model on the Monitors through an existing distributed security kernel—the Trusted Timely Computing Base (TTCB) [12].

The Monitor system is implemented and deployed on our university's campuswide network. It is used to provide *detection* and *diagnosis* functionality to a streaming video application running over a reliable multicast application called TRAM [9]. The latency and accuracy of diagnosis are measured by using fault injection experiments. The Monitor accuracy is found to decrease with an increasing data rate using a pessimistic version of the matching algorithm. The pessimistic version performs matching of all observed messages at the Monitor and is targeted at environments with high failure rates. In contrast, the optimistic version of the protocol only performs matching when a failure is detected. Switching to an optimistic version gives an improved diagnosis accuracy of 85 percent at 175 Kbyte/sec compared to 63 percent in the pessimistic case, but this comes at the cost of higher diagnosis latency.

The paper makes the following contributions:

1. It provides a distributed protocol for an accurate diagnosis of failures. The diagnosis protocol is optimal among algorithms in its class, where the class is defined by the amount of information used by the diagnosis algorithm.
2. It maintains a useful abstraction of the observer and the observed systems, with nonintrusive interactions between the two.
3. Diagnosis can be achieved in the presence of failures in the Monitor framework itself and of error propagation across the entire payload system.
4. The system's performance and fault tolerance are demonstrated on a real-world third-party application.

The rest of the paper is organized as follows: Section 2 presents the diagnosis protocol for the PEs, assuming failure-free Monitors. Section 3 deals with Monitor failures. Section 4 presents the analysis of diagnosis accuracy. Section 5 discusses the implementation, experiments, and results. Section 6 reviews related work, and Section 7 concludes the paper.

2 Diagnosing Failures

This section details the diagnosis protocol that is executed in the system to determine the cause of the failure. We assume in this section that a diagnosis is performed by a failure-free Monitor hierarchy verifying the PEs, but we explain in Section 3 how a diagnosis is handled in case of failures in Monitors.

2.1 System Model

The Monitor employs a stateful model for rule matching to perform detection and diagnosis, implying that it maintains the state that persists across messages. It contains a *rule base* consisting of *combinatorial* rules (valid for all points in time in the lifetime of the application) and/or *temporal* rules (valid for limited time periods). The Monitor observes only the external messages of the PEs. It can be placed anywhere in the infrastructure but is typically not cohosted with the PEs to avoid performance impact to the payload system. The desire to have low latency of detection and diagnosis suggests the placement of the Monitor in the vicinity of the PEs. The Monitor architecture consists of the *Data Capturer*, *Rule Matching Engine*, *State Maintainer*, *Decision Maker*, and, finally, the *Diagnosis Engine* to perform diagnosis. The *Data Capturer* snoops over the communication medium to obtain messages. It can be implemented using *active forwarding* by the PEs to the Monitor or by a *passive snooping* mechanism. In passive snooping, the Monitor captures the communication over the channel without any cooperation from the PEs, for example, through the promiscuous mode in a local area network (LAN) or using router support. In the active forwarding mode, the PEs (or an agent resident on the same host) forward each message to the overseeing Monitor. The message exchanges correspond to the events in the rule base of the Monitor. The *Rule Matching* engine is used to match the incoming events with the rules in the rule base. The *State Maintainer* maintains the state-transition diagram (SD) and the current state of each verified PE. Finally, the *Decision Maker* is responsible for making decisions based on the outcome from the Rule Matching Engine. The *Diagnosis Engine* is triggered when a failure is detected, and it uses state information from the State Maintainer to make diagnosis decisions. The previous Monitor architecture in [4] has been extended to add the diagnosis functionality.

The system is comprised of multiple Monitors logically organized into the LM, IM, and Global Monitor (GM). The *LMs* directly verify the PEs. An *IM* collects information from several LMs. An LM filters and sends only aggregate information to the IM. There may be multiple levels of IMs, depending on the number of PEs, their geographical dispersion, and the capacity of the host on which an IM executes. There is a single GM, which only verifies the overall properties of the network. An example of the hierarchical setup with a single level of IM used in our experiments is shown in Fig. 3. The Monitor's functionality of detection and diagnosis is asynchronous to the protocol. Each Monitor maintains a local *logical clock (LC)* for each PE that it is verifying, which it updates at each observable event (send or receive) for that PE (similar to the Lamport clock [37]).

We assume that PEs can exhibit arbitrary failures. The Monitor is capable of handling a varied set of failures. These encompass any fault, which manifests in a failure that violates a rule in the rule base (described in Section 2.2.4). Specifically, the three categories of failures are correctness failures, violation of performance guarantees, and violation of security guarantees. We do not handle collusion at the PE nodes to prevent detection at the Monitor and coding faults at the Monitor. Errors can propagate from one PE to another through the messages that are exchanged between them. Failures in the PEs are detected by the Monitor infrastructure by comparing the observed message exchanges against the *normal* rule base, as opposed to the *strict rule base* (SRB) used during diagnosis (Section 2.2.4). An anomaly in the behavior of the PEs detected by flagging of a rule triggers the diagnostic procedure. We assume that jitter on PE $\rightarrow$ Monitor link is bounded by *phase* (Δt). We further explain in Section 2.2.2 the need for such an assumption. It is important to note that this assumption is weaker than a complete synchrony.

2.2 Diagnosis Protocol

Diagnosis in a distributed manner based on observing only external message exchanges poses significant challenges. It is essential to consider the phenomenon of propagated errors to avoid penalizing a correct node, in which the failure first manifested as a deviation from the normal protocol behavior. As the Monitor has access only to external message exchanges and not to the internal state, the diagnosis must be based on these messages alone. In other words, the Monitor does not have perfect observability of the payload system's state. The PEs may lie within the domains of different LMs. In such cases, the diagnosis is a distributed effort spanning multiple Monitors at different levels (Local, Intermediate, and Global). In order to identify the faulty PE from among a set of suspect PEs, each PE is subjected to a *test* procedure. Since the Monitor treats PEs as blackboxes, it is thus unaware of the valid request response for the protocol and cannot send any explicit *test* message to the PEs. Moreover, the PE may not currently be in the same state as the one in which the fault was triggered. A failure manifested at the PE could be because of a fault that originated at this PE or because of error propagation through a message that the PE received. If the error is propagated through a message, then it must *causally* precede the message that resulted in failure detection. Given that an entity emits message m_2 on the receipt of message m_1, we define causality as $m_1 \rightarrow m_2$, which is the same definition used in distributed checkpointing [44].

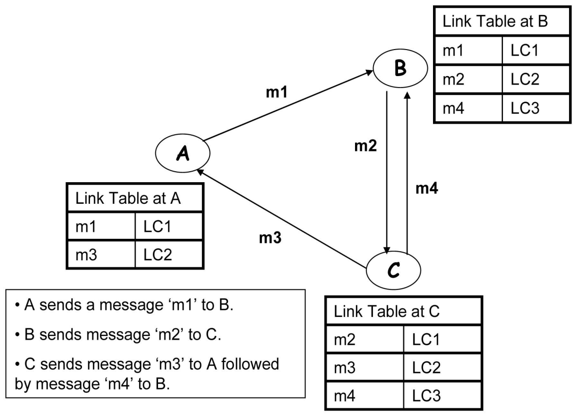

m1
m2
m3
m4
A
B
C
Link Table at A
m1 LC1
m3 LC2
Link Table at B
m1 LC1
m2 LC2
m4 LC3
Link Table at C
m2 LC1
m3 LC2
m4 LC3
• A sends a message 'm1' to B.
• B sends message 'm2' to C.
• C sends message 'm3' to A followed by message 'm4' to B.

Fig. 1. A sample causal graph.

2.2.1 Causal Graph

The *causal graph* captures the causal relationship between messages that are sent and received by PEs, as observed by the Monitor. The causal graph is updated during the normal operation of the protocol. A causal graph at a Monitor m is denoted by $\mathrm{CG_m}$ and is a graph (V, E), where 1) V contains all the PEs verified by m and 2) an edge e contained in E between vertices $v1$ and $v2$ (which represent PEs) indicates the interaction between $v1$ and $v2$ and contains the state about all observed message exchanges between the corresponding PEs, including the LC at each end. We thus establish a correspondence between a PE in the payload system and a node in the causal graph. Henceforth, we use the term "detect a node" to mean detecting a failure in the PE corresponding to the node. The edges are directed and are stored separately as incoming and outgoing with respect to a given node. The edges shall be referred to as *links* from now on. The links are also time-stamped with the local (physical) time at the Monitor, at which the link is created. An example of a causal graph is given in Fig. 1 for the sequence of events described on the lower left corner.

For example in the Link Table for node C, message "4" is assigned an LC time 3. Message $m3$ is causally preceded by message $m2$, which is causally preceded by message $m1$. The messages may be received in a different order at the Monitor because of the asynchronous nature of links. The causality that the Monitor infers is based on the local ordering of messages, as seen by the Monitor observing the messages.

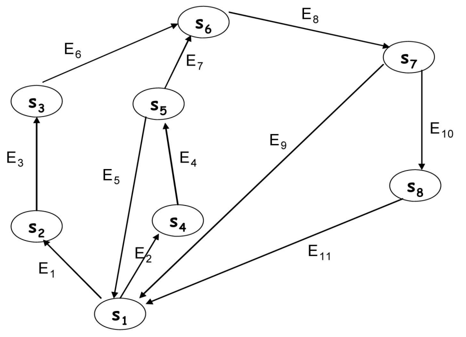


Fig. 2. Sample SD for a $\mathrm{PE}\ \mathrm{P}_1$: illustration of *Protocol Cycle*.

The Monitor may overestimate the causality because if an entity emits $m2$ after receiving $m1$, this does not imply from the application's perspective that $m1$ is dependent on $m2$. However, this latter kind of causality can only be determined by a middleware (such as the Monitor) with application hints.

If, in some applications, this causality information is known a priori, then one can employ user-input methods to modify the causal graph. This view of causality follows the general principles laid down in distributed systems [45], [46], which agree that causality can be characterized using vector time or Lamport time.

2.2.2 Cycle and Phase

In modern distributed protocols, with thousands of communicating PEs, testing all the causally preceding messages is not feasible. We define a time window, over which the diagnosis protocol tests nodes. This time window is called a *Protocol Cycle* to differentiate it from a graph theoretic cycle in the causal graph. The start point of the *Protocol Cycle* (see Fig. 2) denotes how far the diagnosis algorithm should go in history to detect faulty nodes. (Henceforth, if there is no scope for confusion, we use the term cycle as shorthand for protocol cycle.) Cycle boundaries can be decided by using either the SD of the application or the error latency of the application in actual physical time or logical time. First, we present the definition by using the SD.

In the Monitor design, a transition from one state to the next state depends solely on the current state and the event that occurs in the current state. Let there be n PEs verified by the Monitor infrastructure. A reduced SD is maintained at the LM for every verified PE_k, denoted SD_k. Owing to the reduced and finite nature of the SD, it can be assumed that there are repetitions in the set of states traversed by a PE over a long-enough time interval.

There could be several possible runs of different durations for a given PE, each corresponding to a complete task (transaction) as defined in the protocol, for example, a complete round of data and ACK exchange.

Let $\mathrm{S}_{1\mathrm{k}}$ denote the starting state of the PE_k being verified. At an arbitrary starting time t_0, the states of the n PEs would be $\vec{S}_{init} = \{S_{11}, S_{12}, S_{13}, \ldots, S_{1n}\}$. We define *protocol cycle* as the completion of all the possible runs starting from $\vec{S}_{init}$. Each *protocol cycle* will encapsulate several graph cycles, each of which includes the start state of the particular PE. Finding a protocol cycle is NP-complete, since the known

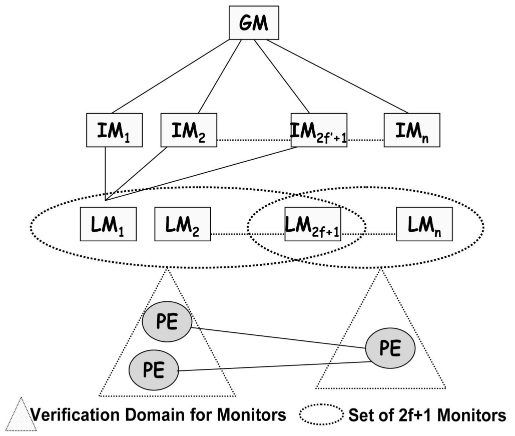


Fig. 3. Redundancy in the Monitor hierarchy.

NP-complete problem of finding the Hamiltonian cycle can be reduced to it in polynomial time.

When a failure is detected in protocol cycle C_i, the checking has to be done till the beginning of C_{i-1} for a deterministic bug. The model for the deterministic bug is that if it manifests itself in state S_{ij} on the receipt of event E_k for PE_i, then it *must* manifest itself every time PE_i goes through the same state and event. For a nondeterministic Heisenbug, the determination may have to go back to further cycle boundaries, since by definition, a nondeterministic bug may not manifest itself repeatedly under the same conditions (same state and event). Alternate strategies may be needed if the number of states to be examined becomes too large through this approach. Then, we can use the upper bound on the error detection latency in the system (for example, as given through analysis in [22]) to come up with the cycle boundary. If we can provide a bound that any error in the application will manifest in time δ, we can limit the messages that need to be checked for errors as being no farther back in (physical) time than δ. If proactive recovery measures such as periodic rebooting [38] are used, then the time points at which the proactive recovery is performed can be taken as cycle boundaries. This is motivated by the claim that latent errors are eliminated at the proactive recovery points.

Let us consider two links in the causal graph L that have been time-stamped with logical times t_{L1} and t_{L2} by the Monitor. Given $t_{L2} > t_{L1}$, we cannot conclude anything about the actual order of these events, since the system is asynchronous. Instead of the synchrony assumption, consider the following more relaxed assumption. Consider that a Monitor M is verifying two PEs: sender S and receiver R. The assumption required by the diagnosis protocol is that the variation in the latency on the $S-M$ channel as well as the variation in the sum of the latency in the $S-R$ and $R-M$ channels is going to be less than a constant Δt, called the *phase*, which is known a priori. If messages M_1 and M_2, corresponding to the two send events at S, are received at Monitor M_1 at (logical) times t_1 and t_2, it is guaranteed that send event M_1 happened before M_2 if $t_{L2} \geq t_{L1} + \Delta t$. This assumption is weaker than the synchrony assumption because it is not dependent on absolute delays on the links; rather, it is dependent on the jitter on the two links. Also, the jitter requirement is in terms of logical time rather than clock time, which is less stringent, since one tick of the LC corresponds to multiple ticks of the physical clock.

2.2.3 Suspicion Set

Flagging of a rule corresponding to a PE represented by node N in the causal graph indicates a failure F and starts the diagnostic procedure. Henceforth, we will use the expression "failure at node N" for a failure detected at the PE corresponding to the causal graph node N. Diagnosis starts at the node where the rule is initially flagged, proceeding to other nodes *suspected* for the failure at node N. All such nodes, along with the link information (that is, the state and event type), form a *Suspicion Set* for failure F at node N, denoted as $\mathrm{SS_{FN}}$.

The suspicion set of a node N consists of all the nodes that have sent their messages in the past, denoted by $\mathrm{SS_N}$. If a failure is detected at node N, then initially $\mathrm{SS_{FN}} = \{\mathrm{SS_N}\}$. Let $\mathrm{SS_N}$ consist of nodes $\{n_1, n_2 \ldots, n_k\}$. Each of the nodes in $\mathrm{SS_{FN}}$ is tested using a test procedure, which is discussed in Section 2.2.4. If a node $n_i \in \mathrm{SS_{FN}}$ is found to be fault-free, then it is removed from the suspicion set, resulting in contraction of the suspicion set. If none of the nodes is found to be faulty, then in the next iteration, the suspicion set for the failure F is expanded to include the suspicion set of all the nodes that existed in $\mathrm{SS_N}$ in the previous iteration. Thus, in the next iteration, $\mathrm{SS_{FN}} = \{\mathrm{SS_{n_1}}, \mathrm{SS_{n_2}} \ldots, \mathrm{SS_{n_k}}\}$. Arriving at the set of nodes that have sent messages to N in this time window is done from the causal graph. Consider that the packet that triggered the diagnosis is sent by N at time τ_S. Then, all the senders of all incoming links into node N, with time stamp t satisfying $C \le t \le \tau_S + \Delta t$, are added to the suspicion list, where Δt is the phase parameter, and C is the cycle boundary. The procedure of contracting and expanding the Suspicion Set repeats recursively until the faulty node is identified, or the cycle boundary is reached, thereby terminating the diagnosis.

2.2.4 Test Procedure

We define the test procedure for a PE to be a set of rules to be matched based on the state of the PE, as maintained in the causal graph. This set of rules constitutes the *strict rule base*, and like the *normal rule base*, which is used for error detection, this consists of temporal and combinatorial rules for expected patterns of message exchanges. The SRB is based on the intuition that a violation does not deterministically lead to a violation of the protocol correctness and, in many cases, gets masked. However, in the case of a fault being manifested through the violation of a rule in the normal rule base as a failure, a violation of a rule in the SRB is regarded as a contributory factor. The strict rules are of the form

$$< Type >< State_1 >< Event_1 >< Count_1 >$$
$$< State_2 >< Event_2 >< Count_2 >,$$

where $(State_1, Event_1, Count_1)$ forms the precondition to be matched, whereas $(State_2, Event_2, Count_2)$ forms the post-condition that should be satisfied for the node to be deemed *not* faulty. The SRB of form $<$ state S, event E, count $C >$ refers to the fact that the event E should have been detected in the state S at least count C number of times. Note that a PE may appear multiple times in the Suspicion Set, for example, in different states, and may be checked multiple times during the diagnostic procedure. Also, the tests are run on the state maintained at the Monitor, without involving the PE, thus satisfying the design goal of nonintrusiveness. Model-based testing discusses several methods of forming these rules automatically. The authors in [42], [43] discuss mechanisms for automatic rule derivations from formal models like the Unified Modeling Language (UML). However, rules for verifying the QoS constraints or vulnerabilities that need to be detected will be specified by the administrator. Thus, rules can be framed through a mix of automated and manual means.

When an SRB rule is used to test a given link l_i in the causal graph, it uses as the phase as preconditions and postconditions in the rule events over a logical window of $\pm\Delta t$, which is measured from the logical time of l_i. This is attributed to the assumption of jitter bound on the communication link, that is, that a message at the Monitor cannot arrive out of order with respect to another message more than Δt away, which originated at the same PE. Each rule in the SRB has some *coverage* to verify a particular PE because it only tests a specific state and event. Therefore, a message sent by an entity in the Suspicion Set must be tested by running multiple rules from the SRB on it. The diagnosis is therefore probabilistic according to the traditional definition [19]. However the PEs are *deterministically diagnosed* as faulty or correct. We develop an analytical model on these assumptions in Section 4.

Like the normal rule base, the rules in the SRB are dependent on the state and the event of the link, but the number of rules is typically much larger than that in the normal rule base. Hence, it is conceivable that the system administrator would not tolerate the overhead of checking against the SRB during a normal protocol operation. A new diagnostic procedure is started for every rule that is flagged at the Monitor. Multiple faults manifesting nearly concurrently would result in multiple rules being flagged, leading to separate and independent diagnostic procedures for each of them. These rule types may not be expressive enough to cover all possible misbehavior in the PEs. Also, rules that depend on the state being aggregated across multiple messages may not be matched under periods of heavy data rate when the Monitor may be overloaded.

2.2.5 Diagnosis Protocol: Flow

This section illustrates the flow of control of the diagnosis protocol and the interactions in the Monitor infrastructure to arrive at a correct diagnosis. We illustrate the set of steps for a failure at a single PE. The protocol for distributed diagnosis among the Monitors comes into play when a suspect node identified by an LM lies outside its domain; that is, the PE required to be tested is not verified by *this* LM. The LM does not contain causal graph information for the suspect node and, hence, requests the corresponding LM verifying the suspect node to carry out the test (step 4):

1. A failure F at PE N is detected by the LM LM_i verifying it.

2. LM_i constructs the suspicion set $\mathrm{SS_N}$ for the failure and adds it to $\mathrm{SS_{FN}}$.
3. For every $N' \in \mathrm{SS_N}$ that belongs to the domain of LM_1, LM_1 tests N' for correctness for the suspect link L' by using rules from the SRB for that particular event and state. If N' is not faulty, then it is removed from $\mathrm{SS_N}$, and $\mathrm{SS_{N'}}$ is added to the $\mathrm{SS_{FN}}$ queue.
4. For every N'' belonging to $\mathrm{SS_N}$ that is not under the domain of LM_i but is under the domain of another Monitor LM_j, LM_i sends a *test request* for N'' and faulty link L'' recursively to higher level Monitors till a common parent for LM_i and LM_j is found, which routes it to LM_j. LM_j tests N'' and sends the result of the test back to LM_i through the same route. If N'' is not faulty, then LM_j also sends the suspicion set corresponding to link L'' for N''.
5. The diagnostic procedure repeats recursively till a node is diagnosed as faulty or till the cycle boundary is reached. In the first case, the node corresponding to which the link is diagnosed as faulty due to violation of rules in the SRB is considered to be faulty. In the latter case, the diagnostic procedure terminates unsuccessfully.

3 Diagnosis in the Presence of Faulty Monitors

An external fault-free "oracle" performing detection and diagnosis, although desirable, is not realistic. In our framework, the Monitors are also considered susceptible to faults. The goal of this section is to show how the diagnosis of faulty PEs can be carried out in the face of arbitrary failures of the Monitors. We assume that Monitors are susceptible to *runtime* failures (for example, due to synchronization errors). In our design, faults in the Monitors are not diagnosed but masked.

3.1 Faults at Local Monitors

If an LM is faulty, then it may exhibit an arbitrary behavior by sending false alarms to higher level Monitors or may drop a valid alarm. In such scenarios, an LM cannot be allowed to perform the diagnostic procedure. We use replication to mask failures at the LMs by allowing multiple LMs to verify a PE. Assuming that there can be failures on up to f LMs, each PE is verified by $2f+1$ LMs, called the Collaborative LM Set (denoted $\mathrm{CS_{LM}}$). An IM can accept that there is an error in the PE being monitored if it receives $f+1$ identical alarms from the LMs in a $\mathrm{CS_{LM}}$. Note that if there is a set of entities (the LMs) whose responses are "voted on" by a fault-free "oracle" (the IM), then only $2f+1$ entities are required under the Byzantine fault model. The communication between LMs and IMs is authenticated to avoid multiple alarms being sent by the same LM. Although all the LMs in the $\mathrm{CS_{LM}}$ verify the same PE, they are spatially disjoint, leading to possibly different views of the state of the PE.

However, for our system, we need to have all correct LMs in a $\mathrm{CS_{LM}}$ agree on the failure alarms that they send to the IM. Another requirement is defining an order among the alarms sent out by the LMs in a $\mathrm{CS_{LM}}$. The solution to both issues is based on an *atomic-order or total-order multicast protocol* (see the definition in [10]).

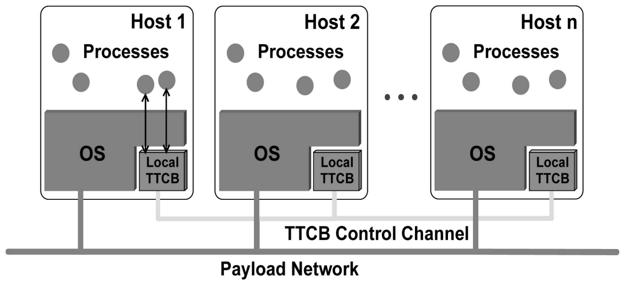


Fig. 4. Architecture of n hosts with a TTCB.

This problem is known to be equivalent to consensus [15], which requires a minimum of $3f+1$ process replicas to be solvable in asynchronous systems with Byzantine faults [11]. We reduce this number of LM replicas to $2f+1$ by using an existing method called the architectural-hybrid fault model [14] (Section 3.3.1).

The algorithm used by the LMs in a $\mathrm{CS_{LM}}$ to agree in an alarm is the following. When the Monitor is initialized, each LM starts a counter with 0. When a rule in an LM raises an alarm, it atomically multicasts that alarm to all LMs in $\mathrm{CS_{LM}}$ (including itself). When the atomic multicast delivers an LM the $(f+1)\mathrm{th}$ copy of the same alarm sent by different LMs in $\mathrm{CS_{LM}}$, it gives that alarm the number indicated by the counter, increases the counter, and sends the message to the IM. It guarantees that all correct LMs agree on the same alarms with a unique order number, ensuring an atomic order. Therefore, the algorithm guarantees that an IM receives identical alarms from all correct LMs verifying a PE.

3.1.1 TTCB and Architectural-Hybrid Fault Model

In this paper, we use the *architectural-hybrid fault model* provided by a distributed security kernel called the TTCB. The notion of architectural-hybrid fault model is simple: We assume different fault models for different parts of the system. Specifically, we assume not only that most of the system can fail arbitrarily or in a Byzantine manner but also that there is a *distributed security kernel* in the system (the TTCB) that can only fail by crashing [12]. The TTCB can be considered a "hard-core" component that provides a small set of secure services such as a Byzantine-resilient consensus to a collection of external entities like the LMs. These entities communicate in a world full of threats, and some of them may even be malicious and try to cheat, but the TTCB is an "oracle" that correct entities can trust and use for the efficient execution of their protocol.

The design and implementation of the TTCB was discussed at length in [13] and, here, we give a brief overview relevant to its application in the Monitor system.

The local TTCB components are connected using a dedicated channel (Fig. 4). The local TTCBs can be protected by being inside some kind of software secure compartment or hardware appliance like a security coprocessor. The security of the control channel can be guaranteed using a private LAN.

3.1.2 Atomic Multicast Protocol

The atomic multicast primitive provides the following properties: 1) All correct recipients deliver the same messages. 2) If the sender is correct, then the correct recipients deliver the sender's message. 3) All messages are delivered in the same order by all correct recipients. The Byzantine-resilient atomic multicast that is tolerant to f out of $2f+1$ faulty replicas is presented in detail in [14]. Here, we describe briefly how it is applied to the Monitor system. Notice that only the nodes with LMs need to have a local TTCB, not the nodes with IMs or the GM. The reason is that the local TTCBs at the different entities need to be connected through a dedicated control channel. Although it may be feasible to connect the LMs monitoring a specific PE cluster, which are likely to be geographically closely placed, through such a control channel, it is unwieldy for IMs that are unlikely to have geographical proximity.

The core of the solution that we use is one of the simple services provided by the TTCB, that is, the Trusted Multicast Ordering (TMO) [12]. Being a TTCB service, its code lies inside the local TTCBs, and its communication goes in the TTCB control channel. When an LM wants to atomically multicast a message M, it gives the TMO a *hash* of M obtained using a *cryptographic hash function*, for example, SHA-2. When an LM receives a message M, it also gives the TMO a hash of the message. Notice that the messages are sent through the normal payload network, that is, outside the TTCB. However, these channels guarantee the authenticity and integrity of the messages. These channels could be implemented using the Secure Sockets Layer (SSL) or Transport Layer Security (TLS). Finally, when the TTCB has information that f LMs received M, it gives M and all LMs in $\mathrm{CS_{LM}}$ the next order number.

3.2 Faults at the Intermediate Monitors

Next, we augment the model to allow IM failures by having a redundant number of IMs. To tolerate f' faults at the IM level, at least $2f'+1$ IM replicas must be used. Therefore, all LMs in a Collaborative LM Set ($\mathrm{CS_{LM}}$) send alarms to all IMs in a Collaborative IM Set, denoted by $\mathrm{CS_{IM}}$. The output of replicas is voted on by a simple voter (GM in our case). The simplicity of the GM and the fact that it is not distributed makes it reasonable to assume that efforts can reasonably be made to make it fault free. Secure coding methodologies, which are based on a formal verification and static code analysis, can be used to build a fault-free GM. The possibility of faults in Monitors forces an LM in $\mathrm{CS_{LM}}$ to accept a test request only if it receives $f+1$ identical test requests from Monitors in $\mathrm{CS_{IM}}$. An alternative design choice would be to control the entire diagnosis protocol from the lower level (failure prone) Monitors through the use of consensus. This was considered to have unacceptable overhead in a number of messages and rounds for consensus, which would be required for every member of the suspicion set. Also, if the suspicion set spans boundaries of the LM, IMs would anyway be needed for distributed diagnosis.

3.3 Flow of Control of Diagnosis with Failing Monitors

Assume that $\mathrm{CS_{IM}}$ initiates the diagnosis:

1. Failure F at PE N is detected by the $\mathrm{CS_{LM}}$ verifying it, which constructs the suspicion set $\mathrm{SS_N}$ and adds it to $\mathrm{SS_{FN}}$.
2. The LMs assign an order to the alarm by using the atomic multicast protocol and send an alarm, along with $\mathrm{SS_{FN}}$, up to all the IMs in $\mathrm{CS_{IM}}$.
3. The IMs wait for $f+1$ identical alarms and then start the diagnostic procedure.
4. For every $N' \in \mathrm{SS_{FN}}$, the (correct) IMs in a $\mathrm{CS_{IM}}$ send a test request to the $\mathrm{CS_{LM}}$ to verify N'.
5. Each $LM \in \mathrm{CS_{LM}}$ that receives $f+1$ identical test requests from IMs in $\mathrm{CS_{IM}}$ tests N' for the correctness of the suspect link L' by using the SRB.
6. The test results are sent above to the IMs in $\mathrm{CS_{IM}}$, who vote on the $f+1$ identical responses to decide if N' is faulty. If N' is not faulty, then it is removed from $\mathrm{SS_N}$, and $\mathrm{SS_{N'}}$ is added to $\mathrm{SS_{FN}}$.
7. If a PE N'' lies outside the verification domain of the IMs in $\mathrm{CS_{IM}}$, then a *test requests* for N'', and faulty link L'' is sent recursively to higher level Monitors, which send the request down the tree to the relevant set of LMs verifying N''. The result of the test is sent back to the IMs through the same route. If N'' is not faulty, then the corresponding suspicion set is also sent along.
8. The diagnostic procedure repeats recursively until a node is diagnosed as faulty or until the cycle boundary is reached.

4 Analysis of Diagnosis Accuracy

For easier understanding and comparison, we follow a similar notation to that in [23]. Consider a k-regular directed graph, with a node representing a PE and an edge representing the message exchange between the PEs. A node is faulty with probability λ. An error can propagate through a message sent by the node with probability ρ, given that the node is faulty. The probability of error propagation through the message is $\rho\lambda$. An error in the node can be caused by a fault in the node or due to an error propagated through one of the incoming links. A test executed on the node has a fault detection coverage c_i if the node n_i is faulty (that is, the probability of detecting a faulty node is c_i) and a coverage d_i if the node has an error that has propagated from some incoming links. For an ideal test, $c_i = 1$, and $d_i = 1$. Let c and d be the average values for the detection coverage for the fault and propagated error over all nodes. Let the number of tests from the SRB performed on the node be T and the total number of nodes be N. Each test yields an output $\mathrm{O} \in \{0,1\}$, where an output 0 means that the node passes the test, and output 1 means that it fails the test. Assume that a node is determined to be faulty if there are z or more ones in the total number of tests $z \in (0,T)$. Let π be the event that a node is faulty and π' be the complement event. Based on the model

$$\mathrm{A} = \mathrm{Prob}(\mathrm{test} = 1|\pi) = c, \tag{1}$$

$$\mathrm{B} = \mathrm{Prob}(\mathrm{test} = 1|\pi') = d(1-(1-\rho\lambda)^k), \tag{2}$$

$$\begin{aligned}&\mathrm{Prob}(z - ones|\pi) = C(T,z)A^z\,(1-A)^{T-z}\\&(\text{where } C \text{ is the binomial coefficient}),\end{aligned} \tag{3}$$

$$Prob(z - ones|\pi') = C(T,z)B^z\,(1-B)^{T-z}. \tag{4}$$

One figure of merit for the diagnostic process is the probability of detecting the original faulty node causing the failure. The *posterior* probability is given by

$$Prob(\pi|z-ones) = Prob(z-ones|\pi). \\ Prob(\pi)/Prob(z-ones),$$

where $Prob(\mathrm{z}-ones)$ is given by $(3).\lambda + (4).(1-\lambda)$ using the total probability formula:

$$= \frac{\lambda\, C(T,z)\, A^z (1-A)^{T-z}}{\lambda\, C(T,z)\, A^z(1-A)^{T-z} + (1-\lambda)\, C(T,z)\, B^z(1-B)^{T-z}} \\ = \frac{1}{1+\frac{(1-\lambda)}{\lambda}\left(\frac{B}{A}\right)^z\left(\frac{1-B}{1-A}\right)^{T-z}}. \tag{5}$$

Equation (5) matches with the one derived by Fussel and Rangarajan (FR) [19], with the following mapping: R (number of rounds) there maps to T here, since in each round of the FR algorithm, the same test is performed.

Now, consider B from (2):

$$\mathrm{B} = d(1-(1-\rho\lambda)^k),$$

taking the number of messages to be very large, we can assume that $\mathrm{k} \to \infty$ reduces to $d(1-e^{k\rho\lambda})$ because $\rho\lambda \to 0$. We can rewrite (5) as

$$\mathrm{Prob}(\pi|z-ones) = 1/1 + \mathrm{F(z)},$$
$$\text{where } \mathrm{F(z)} = ((1-\lambda)/\lambda).(\mathrm{B/A})^z.((1-\mathrm{B})/(1-\mathrm{A}))^{T-z}.$$

We claim that (5) is a monotonically increasing function of z. Note that A and $\mathrm{B} \in (0,1)$. Also, for realistic situations, the probability of a node being faulty is much greater than the probability of a propagated error affecting a node (this is a common assumption in the fault tolerance literature [7], [25]). Any reasonable diagnosis test should be able to distinguish between a node being the originator of a fault (high probability of $\pi = 1$) and one that is the victim of a propagated error (low probability of $\pi = 1$). Therefore, A > B. Let us represent $\mathrm{F}(z)$ as $k\beta^z\mu^{T-z}$. For $A > B$, $\beta < 1$, and $\mu > 1$ and, therefore, $\mathrm{Prob}(\pi|z-ones)$ increases with z. This can also be proved through showing $d(Prob(\pi|z-ones))/dz > 0$. This implies that the higher the value of z for a fixed T, the greater the confidence in the diagnostic process. In other words, the diagnostic process is well behaved as per the definition in [23].

Theorem. *The diagnosis algorithm provides an asymptotically correct diagnosis for $N \to \infty$ for $k \geq 2$ and $T \geq \alpha(N)log(N)$, where $\alpha(N) \to \infty$ arbitrarily slowly as $N \to \infty$. It is also optimal in the diagnosis accuracy among the diagnosis algorithms in its class.*

Proof. We observe that the posterior probability given by (5) matches the posterior probability of the FR algorithm [19]. Further, it is proved in [19] that a diagnosis algorithm that produces this form of posterior probability is asymptotically optimal if T is chosen at least as large as $\alpha(N)log(N)$, where $\alpha(N)$ grows arbitrarily slowly as compared to N. Note that tests T can grow sublinearly compared to the number of nodes to satisfy the condition. Thus, if we increase the number of nodes by Δ such that each additional node is similar in nature to one of the existing N nodes, then tests from the existing rule base can be used on the new nodes. For example, scaling the system by increasing the number of receivers would meet this condition. If such a mechanism is followed, then, trivially, the total number of tests grows linearly with the number of nodes and would satisfy the theorem. This property is true for the largest class of distributed applications. There are a small number of distinct types of entities, and for large deployments, the number of entities of a kind increases. Thus, having T grow linearly with N is simply a case of performing the tests on the new entities. Note that the test results are still independent, assuming independent PE failures.

Our algorithm falls in the 3AM (m-threshold local diagnosis) category, as defined in [23], since 1) all testing are done with local knowledge and 2) a threshold number of tests needs to fail for an entity to be diagnosed as faulty. Hence, the algorithm tends to perfect the behavior asymptotically when $k \geq 2$, and T grows as $\alpha(N)log(N)$. Note that our diagnosis algorithm is also asymptotically correct for the asymptotic behavior of T, which is independent of N, since (5) $\lim\limits_{T\to\infty}\lim\limits_{z\to T} Prob(\pi|z-ones)$ approaches 1. Equation (5) is an increasing function of z. Hence, we find the value $z = z_{th}$, which provides $\mathrm{Prob}(\pi|z-ones) = 0.5$, and set the algorithm to conclude that the node is faulty if $z > z_{th}$ and nonfaulty otherwise. Equating (5) to 0.5 and simplifying, we get

$$z_{th} = \frac{\log(\frac{1-\lambda}{\lambda})}{\log\left(\frac{A(1-B)}{(1-A)B}\right)} + T\frac{\log(\frac{1-B}{1-A})}{\log\left(\frac{A(1-B)}{(1-A)B}\right)}.$$

Therefore, using the property of $\mathrm{Prob}(\pi|z-ones)$ being an increasing function of z and [39, Theorem 1], we conclude that our diagnosis algorithm is optimal in its class 3AM. It is important to note that although the models considered in [19] and in this paper are completely different, the equations for a posteriori probability have the same form, making our diagnosis algorithm optimal. □

5 Implementation, Experiments, and Results

The diagnosis protocol implementation is demonstrated by running a streaming video application on top of TRAM. TRAM is a tree-based reliable multicast protocol consisting of a single sender, multiple repair heads (RHs), and receivers [9]. Data is multicast by the sender to the receivers, with RH(s) being responsible for local repairs of lost packets. An ACK message is sent by a receiver after every *ACK window* worth of packets has been received, or an *ACK interval* timer goes off. The RHs aggregate ACKs from all its members and send an aggregate ACK up to the higher level to avoid the problem of ACK implosion. During the start of the session, *beacon* packets are sent by the sender to advertise the session and to invite receivers. Receivers join by using *head bind* (HB) messages and are accepted using *head acknowledge* (HA) messages from the sender or an RH. TRAM entities periodically exchange *hello* messages to detect failures. The Monitor is given the SRB, along with the STD and the normal rule base, as input. An example of a temporal rule in the normal rule base is that the number of data packets observed during a time period of 5,000 ms should be between 30 and 500. The thresholds are calculated using the maximum and minimum data rates required by TRAM, as specified by the user. Another example is that there should not be two *HB* messages sent by a receiver within 500 ms during the data receiving state, as the receiver could be malicious and be frequently switching RHs. An example of a strict rule used in our experiments for the sender is *SR*: *HI S2 E11 1 S2 E9 1*. Here, *SR* stands for a strict rule (as opposed to normal rule base rules used for detection), with *HI* denoting the *hybrid incoming* rule. It signifies that the precondition event is incoming, and the post condition event is outgoing. If in state S2, the receiver has received a data packet (E11), say, with link ID as *d*, then there must be an *ACK* packet (E9) within the phase interval around *d*. This rule ensures that the receiver sends an *ACK* packet upon receiving data packet(s). Another SRB rule bounds the *hello* to be only sent when an entity is in the data-transmission-reception state to prevent a malicious receiver from *hello* flooding. In our experiments, the number of SRB rules to test a link varied from 4 to 8, depending on the state of the link.

5.1 Optimistic and Pessimistic Link Building

There are two approaches to building the causal graph at the Monitor based on the observed messages. Each incoming (outgoing) message to (from) a node is stored in a vector of incoming (outgoing) links for that node. A link ID (logical time stamp) is assigned to the link, along with the physical time, state, and event type. A link contains two IDs: one is for the node that sent it, and another for the receiving node. For this link to be completed in the causal graph, a matching is required between the sending and the receiving PEs' messages. The link A$\rightarrow$ B will be matched once the message sent by A and the corresponding one received by B are seen at the Monitor. Matching all the incoming packets during runtime is referred to as the *pessimistic approach*, and this entails an enormous overhead. This approach results in low diagnosis latency but also results in some links not being matched at runtime due to an overload at the Monitor. This causes a drop in the accuracy of the diagnosis protocol.

Note that the matched links are not used if a failure is not detected in the same cycle. This is leveraged in the *optimistic approach*. In this approach, at runtime, the Monitor simply stores the link in the causal graph and marks it as being unmatched. Link matching is performed when the diagnosis is triggered on failure. This results in less overhead at the Monitor but high storage overhead and diagnosis latency. We perform experiments to give a comparative evaluation of the optimistic and the pessimistic approaches.

5.2 Fault Model and Fault Injection

For exercising the diagnosis protocol, we perform *fault injection* in the header of the TRAM packets transmitted by the sender. It must be noted that the faults are considered to be accidental faults. Malicious nodes launching deliberate attacks on the system are beyond the scope of this paper. The Monitor inspects only the header and is not aware of the payload. Hence, the faults are only injected into the packet header. The fault is *injected* by changing bits in the header after the PE has sent the message. Note that the emulated faults are not simply message errors but are also symptomatic of faults in the protocol itself. For example, a faulty receiver may send a NACK instead of an ACK upon successfully receiving a data packet. Errors in message transmission can indeed be detected by a checksum that is computed on the header. However, the Monitor is responsible for detecting and diagnosing errors in the protocol itself, which are clearly outside the purview of the checksum. As explained previously, the faults at the Monitor level are masked through replication. The strict rules are used to diagnose the faults, with each rule having some coverage. We use the following kinds of injections for a *burst length* period of time:

- *Stuck-At injection (SA)*. For all packets in the burst length, a randomly selected header field value is changed to a random but valid value. This kind of injection mimics multiple protocol errors, where because of a software bug or misconfiguration, a PE sends incorrect packets repeatedly. An important class of security errors that this injection mimics is flooding of packets (like the SYN packet in the TCP SYN attack) instead of protocol-legitimate packets.
- *Directed injection (Dir)*. For each packet, a specific header field is chosen for one experiment and changed to a random but valid value, with different values in different runs.
- *Specific injection (Spec)*. Specs consist of specific protocol errors like slow data rate, dropping ACKs, and hello message flooding. Burst error is chosen as the fault model over a single error, since the protocol is robust enough that single errors are almost always tolerated by built-in mechanisms in the protocol.

5.3 Test Setup and Topology

Fig. 5b illustrates the topology used for the accuracy and the latency experiments on TRAM, with the components distributed over the campus network (henceforth called TRAM-D), whereas Fig. 5a shows the topology for the local deployment of TRAM (TRAM-L). TRAM-D is important, since a real deployment will likely have receivers that are distant from the sender. TRAM-L lets us control the environment and therefore run a more extensive set of tests (for example, with a large range of data rates). The PEs and the LMs are capable of failing, whereas we assume for these experiments that the IMs and the GM are fault free. The sender, the receivers, and the RHs do active forwarding of the packet to the respective LMs. The minimum data rate in TRAM needed to support the quality of the video application is set at 25 Kilobits per second (Kbps). The Monitors are on the same LAN, which is different from the LAN on which the PEs are located. The routers are interconnected through links of 1 gigabit per second (Gbps), and each cluster machine is connected to a router through a link of 100 megabits per second (Mbps).

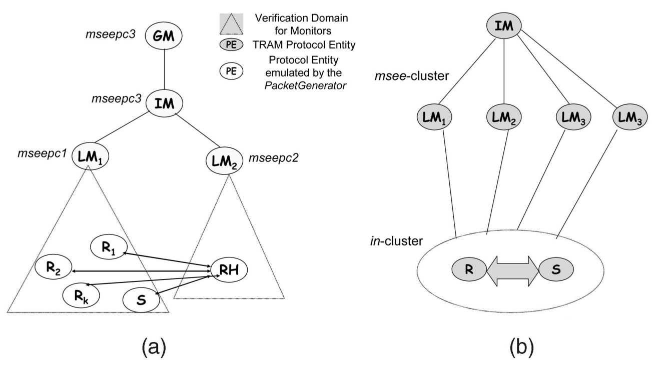


Fig. 5. Topology used for accuracy and latency experiments in (a) TRAM-L and (b) TRAM-D.

5.4 Accuracy and Latency Results for TRAM-L

We measure the accuracy and latency for the diagnosis algorithm on the TRAM protocol through fault injection in the header of sender packets.

Although Fig. 5a depicts multiple receivers and our experiment indeed uses multiple receivers, we monitor (or collect data) for a single receiver receiving packets from an RH, which is connected to the sender. Accuracy is defined as the ratio of the number of correct diagnosis to the total number of diagnosis protocol execution. This definition eliminates any detection inaccuracy from the diagnosis performance. The diagnosis accuracy decreases if the algorithm terminates without diagnosing any node as faulty (*incomplete*) or if it flags a correct node to be faulty (*incorrect*). Latency is defined as the time elapsed between the initiation of diagnosis and diagnosing a node as being faulty, either correctly or incorrectly, or an incomplete termination of the algorithm. We perform experiments with both the optimistic and the pessimistic approaches of link building. There are thus two dimensions to the experiments: the link building approach (abbreviated as *Opt* and *Pes*) and the fault injection strategy (*SA* for Stuck-at, *Dir* for Directed, and *Spec* for Specific). In the interest of space, a representative sample of results is shown. The results are plotted for *Opt-SA*, *Opt-Dir*, and *Pes-Dir*, with a fixed burst length of 300 ms for each injected fault. The interpacket delay is varied to achieve the desired increase in the data rate. Delay d is inserted using a Gaussian random variable with mean d and standard deviation $0.01d$. Each point is averaged over four injections and 20-58 diagnosis instances, depending on the number of detections, which, in turn, depends on the rate of incoming faulty packets.

Fig. 6a shows that for *Pes-Dir*, the accuracy is a monotonically decreasing function with data rate. The diagnosis accuracy drops to as low as 33 percent for data rate at 355 Kbytes/sec. A rate mismatch between the matching of links for causal graph creation (slower process) and the arrival of packets at high data rates (faster process) causes this decrease. The lack of adequate buffer causes packet drops, leading to missing links in the causal graph, which leads to a drop in accuracy. Another factor is the lack of synchronization between the causal graph formation process and the suspicion set creation and testing process. Thus, the latter may be triggered before the former completes, leading to inaccuracies.

For *Opt-Dir*, the accuracy is high for a small data rate but decreases with the increase in data rate. Unlike *Pes-Dir*, here, the accuracy does not drop below 80 percent. The link matching and the causal graph completion are triggered when the diagnosis starts, and the diagnosis algorithm tests the links only after the causal graph is complete, resulting in a higher accuracy compared to *Pes-Dir*. This advantage becomes significant at high data rates. Also, beyond a threshold, further increasing the data rate does not affect the latency because the number of incorrect packets increases, which helps the diagnosis because the current algorithm stops as soon as a single faulty link is identified. The accuracy of *Opt-SA* is slightly lower than that of *Opt-Dir*, since in the former, the same message type is injected for the entire burst. If a rule for the message type does not exist in the SRB, the diagnosis is incomplete.

Fig. 6b shows the scalability results, where the latency and accuracy of diagnosis is observed over an increasing number of receivers. We employ the TRAM-L topology depicted in Fig. 5a, with each receiver receiving data at 40 Kbytes/sec. Thus, the total incoming rate for each receiver at the Monitor is 80 Kbytes/sec, taking both sender and receiver. We can observe that at 40 receivers, the Monitor system gets overwhelmed, causing the latency to increase and the accuracy to drop. The load on the system is CPU intensive, composed of detection and diagnosis, which are being constantly performed on the incoming messages and forming the causal graph, which is performed when diagnosis is done. Beyond the breaking point, the excessive

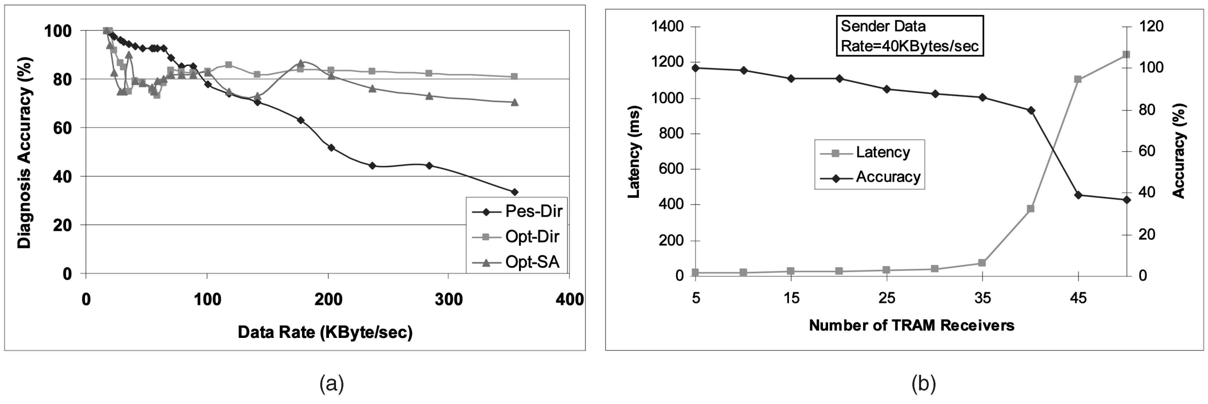
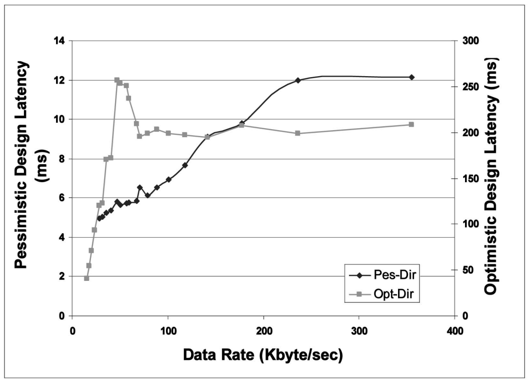
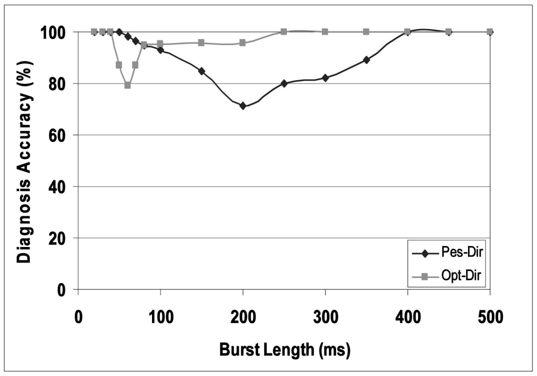


Fig. 6. (a) Variation of diagnosis accuracy with data rate of sender. (b) Latency and accuracy, with an increasing number of receivers.



(a)



(b)

Fig. 7. (a) Variation of latency with data rate. (b) Diagnosis accuracy with burst length for optimistic and pessimistic approaches.

load on the Monitor system causes several diagnoses to be incomplete, leading to a drop in accuracy. Diagnosis instances can be incomplete because of either the IM or the LM being overloaded. If an LM is overloaded, then it is likely that the CG is incomplete, leading to an inaccurate SS, which causes the diagnosis to fail. If an IM is overloaded, then it may drop incoming messages containing the SS from the LMs, leading to incomplete diagnosis.

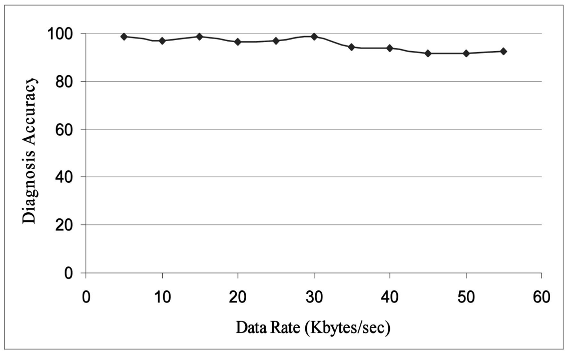
Fig. 7a graphs the latency of diagnosis with an increasing data rate. Notice the significantly higher latency for the optimistic case compared to the pessimistic one. We can see that for the *Pes-Dir* case, the latency increases with the data rate, which is expected because there are more packets to be tested by each rule in the SRB. The latency tends to saturate at high data rates because of an incomplete causal graph, which leads to an inaccurate early termination. On the other hand, in the *Opt-Dir* scenario, the latency keeps increasing with the data rate because of the lazy link matching.

Effect of burst length. We study the impact of burst length on the diagnosis accuracy. We keep the data rate low, that is, at 15 KBytes/sec, to isolate the effects due to high data rate. The diagnosis, as shown in Fig. 7, is accurate for low and high values of burst length. For a small burst length, a small number of incorrect packets get injected, leading to low entropy in the payload system, which is easy to detect. As the burst length increases, more incorrect packets are received by the Monitor, which increases the entropy and, hence, decreases the accuracy. Beyond a certain burst length, more incorrect packets come in, helping in the diagnosis. A more "systems-level" explanation for the increasing part of the curve on the right side is that as the burst length increases, the proportion of SRB rules that match across the boundary of the burst length decreases. These are the SRB rules that are likely to lead to incorrect diagnosis, since they are dealing with a mix of correct and incorrect packets.

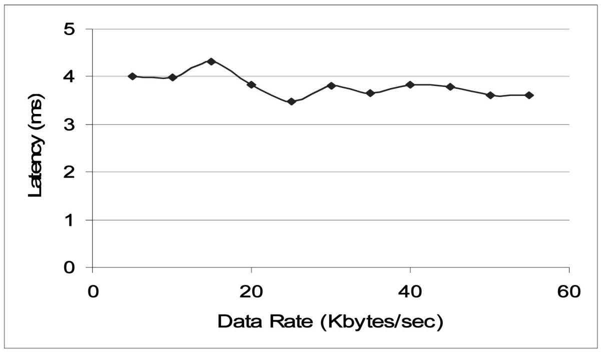
5.5 Accuracy and Latency Results for TRAM-D

In this set of experiments, we measure the accuracy and latency of the pessimistic approach of the diagnosis protocol on TRAM while performing specific fault injection, namely, reducing the data rate from the sender. The latency and accuracy values are averaged over 200 diagnosis instances for each data rate. Fig. 8a shows that the accuracy of diagnosis drops from a high of 98 percent at 15 Kbytes/sec to 91 percent for 50 Kytes/sec. As the data rate increases, the creation of links in the causal graph gets delayed, as incoming packets are pushed off to a buffer for subsequent matching.

If a diagnosis is triggered, which needs to follow one of the missing links, it results in an incomplete diagnosis, leading to a drop in the accuracy. Fig. 8b shows the latency of diagnosis with an increasing data rate. Intuitively, when the data rate increases, increasing the load on the Monitor should cause the latency to increase. However, the data rate used is low enough that it has no significant effect.



(a)



(b)

Fig. 8. (a) Diagnosis accuracy and (b) latency variation with increasing data rate of sender in TRAM-D.

6 Related Work

Different problem. Prior to diagnosis is detection of failures, whether accidental or malicious. There is a plethora of work on failure detection using heartbeats, watchdogs, and Intrusion Detection Systems (IDSs). They differ in the level of intrusiveness with respect to the application entities. Interestingly, the automated response mechanisms associated with many detectors take local responses, assuming that the detection site is the origin of the fault, with no error propagation. This is clearly a leap of faith, as has been shown repeatedly (see [25], [26]). King and Chen proposed building a dependency graph by using logs at the operating system level to track intrusions [40]. This can be looked upon as an alternative way of extracting causal information, which can be incorporated in the Monitor's Causal Graph formation. An extensive hierarchical framework to provide a data management system was proposed through Astrolabe [41]. Renesse et al. provide a dynamic data management system, which is used to maintain aggregate management information base (MIB) information about the underlying system. This is primarily a data management framework, where this collected data could be used for various purposes, including system monitoring. System monitoring has been proposed by several researchers. Our contribution is not in system monitoring, instead in using the monitoring for a distributed diagnosis. In that sense, an existing monitoring technique, if it can work with blackbox participants, can be adapted to work with the Monitor framework.

Different approaches to the same problem. Diagnosis in distributed systems has been an important problem area and was first addressed in a seminal paper by Preparata et al. [17], known as the Preparate-Metze-Chien (PMC) method. The PMC approach, along with several other deterministic models [8], assumes tests to be perfect and mandates that each entity be tested a fixed number of times. The fault model assumed is often restrictive such as permanent failures [21]. Probabilistic diagnosis was first introduced in [18]. Probabilistic diagnosis can only diagnose faulty nodes with a high probability but can relax assumptions about the nature of the fault (intermittent faulty nodes can be diagnosed) and the structure of the testing graph. Follow-up work focused on multiple syndrome testing [19], where multiple syndromes were generated for the same node proceeding in multiple lock steps. Both use the comparison-based testing approach, whereby a test workload is executed by multiple nodes, and a difference indicates suspicion of failure. More recently, Duarte and Nanya [30] propose a fully distributed algorithm that allows every fault-free node to achieve diagnosis in, at most, $(\log N)^2$ testing rounds. All of these approaches are fundamentally different from ours, since the tested and the testing systems are the same, and the explicit tests for diagnosis make the process intrusive to the tested entity.

Similar approach to a different problem. There has been considerable work on diagnosing performance problems in distributed systems. They can be classified into active probing or perturbation and passive monitoring approaches. In the first class, in [27], [28], the authors use, respectively, fault injection and forcible locks on shared objects to determine the location of performance bottlenecks. The second approach uses execution traces for blackbox applications and has similarities to the Monitor approach (see [29], [31], [32]). For example, in [29], the debugging system performs an analysis of message traces to determine the causes of long latencies. However, in all of these work, the goal is not the diagnosis of faults but the deduction of dependencies in distributed systems, which may enable humans to debug performance problems. These may be regarded as point solutions in the broader class of diagnosis problems.

TTCB. There is an abundance of work on consensus. The consensus has been applied to various kinds of environments, with different timing assumptions and types of failures ranging from crash to arbitrary (see [20] for a survey). Our approach of using TTCBs on the Monitor replicas for atomic multicast is derived from the work in [13], [14], which showed how consensus can be achieved in a hybrid failure and communication model system.

7 Conclusions

We have presented a Monitor system for a distributed diagnosis of failures in the PEs in a distributed application. The overall system is structured as a payload system and a Monitor system, each of which may fail in arbitrary ways. The demonstration is given for a streaming video application running on top of a reliable multicast protocol called TRAM. The hierarchical Monitor system is shown to be able to perform diagnosis in the presence of error propagation and using cooperation between the individual Monitor elements. The diagnosis accuracy is higher than 90 percent for the streaming video application under a large range of scenarios. Next, we plan to explore the cooperative testing by multiple Monitors, testing in the face of uncertain information, and effect of placement of the Monitors.

Acknowledgments

This material is based upon work partially supported by the Purdue Research Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Purdue Research Foundation.

References

[1] META Group, "Quantifying Performance Loss: IT Performance Eng. and Measurement Strategies," http://www.metagroup.com/cgi-bin/inetcgi/jsp/displayArticle.do?oid=18750, 2000.

[2] Costs of Computer Downtime to American Businesses, FIND/SVP, 1993.

[3] G. Khanna, J. Rogers, and S. Bagchi, "Failure Handling in a Reliable Multicast Protocol for Improving Buffer Utilization and Accommodating Heterogeneous Receivers," *Proc. 10th IEEE Pacific Rim Dependable Computing Conf. (PRDC '04),* pp. 15-24, 2004.

[4] G. Khanna, P. Varadharajan, and S. Bagchi, "Self-Checking Network Protocols: A Monitor-Based Approach," *Proc. 23rd IEEE Symp. Reliable Distributed Systems (SRDS '04),* pp. 18-30, 2004.

[5] M. Diaz, G. Juanole, and J.-P. Courtiat, "Observer—A Concept for Formal On-Line Validation of Distributed Systems," *IEEE Trans. Software Eng.,* vol. 20, no. 12, pp. 900-913, Dec. 1994.

[6] M. Zulkernine and R.E. Seviora, "A Compositional Approach to Monitoring Distributed Systems," *IEEE Dependable Systems and Networks,* pp. 763-772, 2002.

[7] S. Bagchi, Y. Liu, Z. Kalbarczyk, R.K. Iyer, Y. Levendel, and L. Votta, "A Framework for Database Audit and Control Flow Checking for a Wireless Telephone Network Controller," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '01),* pp. 225-234, 2001.

[8] R. Buskens and R. Bianchini Jr., "Distributed On-Line Diagnosis in the Presence of Arbitrary Faults," *Proc. 23rd Int'l Symp. Fault-Tolerant Computing (FTCS '93)*, 1993.

[9] D.M. Chiu, M. Kadansky, J. Provino, J. Wesley, H. Bischof, and H. Zhu, "A Congestion Control Algorithm for Tree-Based Reliable Multicast Protocols," *Proc. IEEE INFOCOM '02*, pp. 1209-1217, 2002.

[10] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, pp. 225-267, 1996.

[11] G. Bracha and S. Toueg, "Asynchronous Consensus and Broadcast Protocols," *J. ACM*, vol. 32, no. 4, pp. 824-840, 1985.

[12] M. Correia, N.F. Neves, and P. Veríssimo, "How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems," *Proc. 23rd Int'l Symp. Reliable and Distributed Systems (SRDS '04)*, pp. 174-183, 2004.

[13] M. Correia, N.F. Neves, and P. Veríssimo, "The Design of a COTS Real-Time Distributed Security Kernel," *Proc. Fourth European Dependable Computing Conf. (EDCC '02)*, pp. 234-252, 2002.

[14] M. Correia, N.F. Neves, L.C. Lung, and P. Veríssimo, "Low Complexity Byzantine-Resilient Consensus," *Distributed Computing*, vol. 17, no. 3, pp. 237-249, 2005.

[15] A. Mostefaoui, M. Raynal, and C. Travers, "Crash-Resilient Time-Free Eventual Leadership," *Proc. 23rd IEEE Int'l Symp. Reliable Distributed Systems (SRDS '04)*, pp. 208-217, 2004.

[16] I. Katzela and M. Schwartz, "Schemes for Fault Identification in Communication Networks," *IEEE/ACM Trans. Networking*, vol. 3, no. 6, pp. 753-764, 1995.

[17] F.P. Preparata, G. Metze, and R.T. Chien, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Trans. Electronic Computers*, vol. 16, no. 6, pp. 848-854, 1967.

[18] S. Maheshwari and S. Hakimi, "On Models for Diagnosable Systems and Probabilistic Fault Diagnosis," *IEEE Trans. Computers*, vol. 25, pp. 228-236, 1976.

[19] D. Fussel and S. Rangarajan, "Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity," *Proc. 19th Int'l IEEE Symp. Fault-Tolerant Computing (FTCS '89)*, pp. 560-565, 1989.

[20] M. Barborak, A. Dahbura, and M. Malek, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, vol. 25, no. 2, pp. 171-220, June 1993.

[21] A. Bagchi and S. Hakimi, "An Optimal Algorithm for Distributed System Level Diagnosis," *Proc. 21st Int'l Symp. Fault Tolerant Computing (FTCS '91)*, pp. 214-221, 1991.

[22] R. Chillarege and R.K. Iyer, "Measurement-Based Analysis of Error Latency," *IEEE Trans. Computers*, vol. 36, no. 5, May 1987.

[23] S. Lee and K.G. Shin, "On Probabilistic Diagnosis of Multiprocessor Systems Using Multiple Syndromes," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 6, pp. 630-638, June 1994.

[24] A. Avizienis and J.-C. Laprie, "Dependable Computing: From Concepts to Design Diversity," *Proc. IEEE*, vol. 74, no. 5, pp. 629-638, 1986.

[25] S. Chandra and P.M. Chen, "How Fail-Stop Are Faulty Programs?" *Proc. 28th Ann. Int'l Symp. Fault-Tolerant Computing (FTCS '98)*, pp. 240-249, 1998.

[26] H. Madeira and J.G. Silva, "Experimental Evaluation of the Fail-Silent Behavior in Computers without Error Masking," *Proc. 24th Int'l Symp. Fault-Tolerant Computing (FTCS '94)*, pp. 350-359, 1994.

[27] A. Brown, G. Kar, and A. Keller, "An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Environment," *Proc. Int'l Symp. Integrated Network Management (IM '01)*, 2001.

[28] S. Bagchi, G. Kar, and J.L. Hellerstein, "Dependency Analysis in Distributed Systems Using Fault Injection: Application to Problem Determination in an e-Commerce Environment," *Proc. 12th Int'l Workshop Distributed Systems: Operations and Management (DSOM '01)*, 2001.

[29] M.K. Aguilera, J.C. Mogul, J.L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03)*, 2003.

[30] E.P. Duarte and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," *IEEE Trans. Computers*, vol. 47, no. 1, pp. 34-45, Jan. 1998.

[31] J.L. Hellerstein, "A General-Purpose Algorithm for Quantitative Diagnosis of Performance Problems," *J. Network and Systems Management*, 2003.

[32] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: On-Line Modelling and Performance-Aware Systems," *Proc. ACM Ninth Workshop Hot Topics in Operating Systems (HotOS '03)*, pp. 85-90, 2003.

[33] R. Alur, R.K. Brayton, T.A. Henzinger, S. Qadeer, and S.K. Rajamani, "Partial-Order Reduction in Symbolic State-Space Exploration," *Proc. Ninth Int'l Conf. Computer-Aided Verification (CAV '97)*, pp. 340-351, 1997.

[34] K. Ravi and F. Somenzi, "High–Density Reachability Analysis," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '95)*, pp. 154-158, 1995.

[35] J.R. Burch, E.M. Clarke, and D.E. Long, "Symbolic Model Checking with Partitioned Transition Relations," *Proc. Design Automation Conf. (DAC '91)*, pp. 403-407, 1991.

[36] K.L. McMillan, *Symbolic Model Checking: An Approach to the State-Explosion Problem.* Kluwer Academic Publishers, 1993.

[37] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

[38] M. Castro and B. Liskov, "Proactive Recovery in a Byzantine-Fault-Tolerant System," *Proc. Fourth Symp. Operating Systems Design and Implementation (OSDI '00)*, Oct. 2000.

[39] S. Lee and K. Shin, "Optimal and Efficient Probabilistic Distributed Diagnosis Schemes," *IEEE Trans. Computers*, vol. 42, no. 7, pp. 882-886, July 1993.

[40] S.T. King and P.M. Chen, "Backtracking Intrusions," *Proc. Symp. Operating Systems Principles (SOSP)*, Oct. 2003.

[41] R.V. Renesse, K.P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Trans. Computer Systems*, vol. 21, no. 2, pp. 164-206, 2003.

[42] J. Offutt and A. Abdurazik, "Generating Tests from UML Specifications," *Proc. Second Int'l Conf. Unified Modeling Language—Beyond the Standard (UML '99)*, pp. 416-429, 1999.

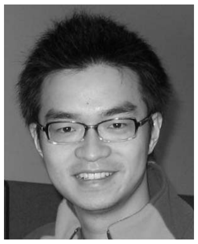
[43] C. Meudec, "Automatic Generation of Software Tests from Formal Specifications," PhD dissertation, The Queen's Univ. of Belfast, 1997.

[44] E.N. Elnozahy, L. Alvisi, Y.M. Wang, and D.B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375-408, Sept. 2002.

[45] R. Schwarz and F. Mattern, "Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail," *Distributed Computing*, vol. 7, no. 3, pp. 149-174, 1994.

[46] O. Babaoglu and K. Marzullo, "Detecting Global States of Distributed System: Fundamental Concepts and Mechanisms," *Distributed Systems*, Addison-Wesley, pp. 55-96, 1993.

Gunjan Khanna received the BS degree from the Indian Institute of Technology, Delhi, in 2002, the master's degree in electrical and computer engineering from Purdue University in December 2003, and the PhD degree from Purdue University in 2007. He currently is working at McKinsey & Company in Pittsburgh. He was an intern at the IBM Research Laboratory in the summers of 2004, 2005, and 2006. His research interests include autonomic detection and diagnosis in distributed systems. He is also exploring fault tolerance in wireless networks. He is a recipient of the Magoon Award for Teaching Excellence in 2005.



Mike Yu Cheng received the BS degree in computer engineering and the MS degree in electrical and computer engineering from Purdue University in 2003 and 2007, respectively. During his graduate studies, he worked on fault-tolerant distributed systems. He is currently with Morningstar in Chicago, Illinois.

Padma Varadharajan received the bachelor's degree from the Birla Institute of Technology and Science, Pilani, India, and the master's degree from the School of Electrical and Computer Engineering, Purdue University, in 2005. Her main research interest is reliability in distributed systems. She is currently a software design engineer at Microsoft Corp. in Redmond, Washington.

Saurabh Bagchi received the MS and PhD degrees from the University of Illinois, Urbana-Champaign, in 1998 and 2001, respectively. He is an assistant professor in the School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana. He is a faculty fellow at the Cyber Center and has a courtesy appointment in the Department of Computer Science, Purdue University. At Purdue, he leads the Dependable Computing Systems Laboratory (DCSL), where he and a set of wildly enthusiastic students try to make and break distributed systems for the good of the world. His work is supported by the US National Science Foundation (NSF), Indiana 21st Century Research and Technology Fund, Avaya, Motorola, and Purdue Research Foundation, with equipment grants from Intel and Motorola. His papers have been runners up for the best paper in the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC '06), 2005 International Conference on Dependable Systems and Networks (DSN), and MTTS 2005. He has been member of the organizing committee and the program committee for DSN and the Symposium on Reliable Distributed Systems (SRDS).

Miguel P. Correia received the PhD degree in computer science from the University of Lisbon in 2003. He is an assistant professor in the Department of Informatics, Faculty of Sciences, University of Lisbon, where he is a member of the Large-Scale Informatics Systems Laboratory (LASIGE) and the Navigators Research Group. He has been involved in several research projects related to intrusion tolerance and security, including the Information Society Technologies—Europe Canada (IST-EC) projects Malicious-and Accidental-Fault Tolerance for Internet Applications (MAFTIA) and CRitical UTility InfrastructurAL Resilience (CRUTIAL), the Resilience for IST Network of Excellence (ReSIST NoE), and national projects. More information about him is available at www.di.fc.ul.pt/~mpc.

Paulo J. Veríssimo is a professor in the Department of Informatics (DI), Faculty of Sciences, University of Lisbon (http://www.di.fc.ul.pt/~pjv), and the director of the Large-Scale Informatics Systems Laboratory (LASIGE, http://lasige.di.fc.ul.pt). He is part of the European Sec and Dep Advisory Board and is an associate editor of the *IEEE Transactions on Dependable and Secure Computing*. He is a former chair of the IEEE Technical Committee on Fault Tolerant Computing and of the steering committee of the International Conference on Dependable Systems and Networks (DSN). He leads the Navigators Research Group, LASIGE. His current research interests include the architecture, middleware, and protocols for distributed, pervasive, and embedded systems in the facets of real-time adaptability and fault/intrusion tolerance. He is the author of more than 130 refereed publications in international scientific conference proceedings and journals in the area and a coauthor of five books. He is a senior member of the IEEE.