Automated Rule-Based Diagnosis Through A Distributed Monitor System

Gunjan Khanna, Mike Yu Cheng, Padma Varadharajan, Saurabh Bagchi
Dependable Computing Systems Lab
School of Electrical & Computer Engineering
Purdue University

Miguel P. Correia, Paulo J. Veríssimo
Navigators Group
Faculty of Sciences
University of Lisboa, Portugal

E-mail: {gkhanna, mikecheng, pvaradha, sbagchi}@purdue.edu, {mpc, pjv}@di.fc.ul.pt

Contact Author: Saurabh Bagchi

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 just detect a failure, but also to diagnose the failure, i.e., 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 black-boxes 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 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 campus-wide network. Fault injection experiments are carried out to evaluate the accuracy and latency of the diagnosis.

Keywords: 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 mis-configurations. The malicious failures may be due to external attackers or internal attackets. The financial consequences of failures to distributed infrastructure can be

gauged from a survey by Meta Group Inc. of 21 industrial sectors in October 2000 [1], which found the mean loss of revenue due to an hour of computer system downtime to be $1.01M. 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 robust infrastructure capable of tolerating the two classes of failures, it is required to provide *detection* and *diagnosis* primitives as part of a fault tolerance infrastructure. Following the definitions in [24], a *fault* is an invalid state or bug underlying in the system, which when triggered becomes an *error*. A *failure* is an external manifestation of an *error* manifested to the end user. A *failure* in a distributed system may be manifested at an entity 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), i.e., 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]. Monitoring and collection of system state in various forms has 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 operate 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 black-box by the Monitor system. While it is possible to build very optimized and specialized fault tolerance mechanisms for specific applications (e.g., like our earlier work in [3] with the 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, the systems are of very large scales with soft real-time guarantees making it particularly challenging to meet all 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 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 the Monitor provides the *detection* and the *diagnosis* functionalities to the PE. The Monitors treat the PEs as black-box and only the causal relation amongst the messages deduced from the send-receive ordering, along with a rule base containing correctness and 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

state that has already been deduced by the Monitors during 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, while 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 distributed diagnosis if the verified PEs lie under the verification domains of different Monitors. We assume 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 campus-wide network. It is used to provide *detection* and *diagnosis* functionality to a streaming video application running over a reliable multicast application called TRAM [9]. Latency and accuracy of diagnosis are measured, using fault injection experiments. The Monitor accuracy is found to decrease with increasing data rate using a pessimistic version of the matching algorithm. The pessmistic 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 failure is detected. Switching to an optimistic version gives improved diagnosis accuracy of 85% at 175 KB/s compared to 63% in the pessimistic case, but comes at the cost of higher diagnosis latency.

The paper makes the following contributions: (1) It provides a distributed protocol for 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 non-intrusive interactions between the two. (3) Diagnosis can be achieved in the presence of failures in the Monitor framework itself and 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 diagnosis is performed by a failure free Monitor hierarchy verifying the PEs but explain in Section 3 how 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 it maintains 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 typically not co-hosted 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 *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, e.g., through the promiscuous mode in a 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 events in the rule base of the Monitor. The *Rule Matching* engine is used to match the incoming events with 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 comprises of multiple Monitors logically organized into Local, Intermediate, and Global Monitors. The *Local Monitors* (LMs) directly verify the PEs. An *Intermediate Monitor* (IM) collects information from several Local Monitors. 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 Global Monitor (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 Figure 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 it is verifying, which it updates at each observable event (send or receive) for that PE (similar to 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 guarantess, 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 which 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* 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 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 internal state, 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 black-boxes 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 which originated at this PE or because of error propagation through a message which the PE received. If the error is propagated through a message then it must *causally* precede the message which 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 as used in distributed checkpointing [44].

2.2.1 Causal Graph

The *causal graph* captures the causal relationship between messages 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 CG_m and is a graph (V, E) where (i) V contains all the PEs verified by *m*; (ii) An edge *e* contained in E, between vertices *v1* and *v2* (which represent PEs) indicates interaction between *v1* and *v2* and contains state about all observed message exchanges between the corresponding PEs including the logical clock (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 detect 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 Figure 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 a logical clock time 3. Message *m3* is causally preceded by message *m2* which is causally preceded by message *m1*.

The messages may be received in 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.

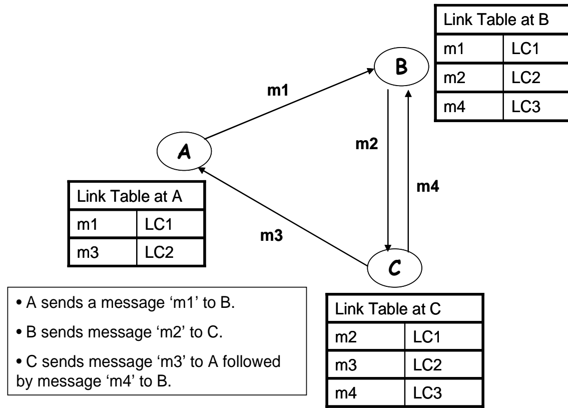


Figure 1: A sample causal graph

The Monitor may over estimate 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 agrees 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 protocol entities, 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* 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 either by using the SD (State-Transition Diagram) of the application

or error latency of the application in actual physical time or logical time. First, we present the definition 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.

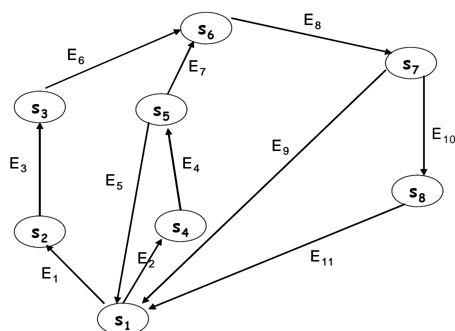


Figure 2: Sample SD for a PE P_1, illustration of Protocol *Cycle*

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, e.g., a complete round of data and ack exchange.

Let S_{1k} 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 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 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 non-deterministic Heisenbug, the determination may have to go

back to further cycle boundaries since by definition, a non-deterministic 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 (e.g., 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 which 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 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 logical clock 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 (i.e. state and event type) form a *Suspicion Set* for failure *F* at node *N* denoted as SS_{FN}.

The suspicion set of a node *N* consists of all the nodes which have sent it messages in the past denoted by SS_N. If a failure is detected at node *N* then initially $SS_{FN}=\{SS_N\}$. Let SS_N consist of nodes $\{n_1, n_2..., n_k\}$. Each of the nodes in SS_{FN} is tested using a test procedure which is discussed in Section 2.2.4. If a node $n_i \in SS_{FN}$ is found to be fault-free then it is removed from the suspicion set resulting in contraction of suspicion set. If none of the nodes is found to be faulty then in the next iteration suspicion set for the failure F is expanded to include the suspicion set of all the nodes which existed in SS_N in the previous iteration. Thus, in the next iteration $SS_{FN} = \{SS_{n_1}, SS_{n_2}..., SS_{n_k}\}$. 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 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 $C \leq t \leq \tau_S + \Delta t$ 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 (SRB)* and like the *normal rule base*, used for error detection, 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, while $(State_2, Event_2, Count_2)$ forms the post-condition that should be satisfied for the node to be deemed *not* faulty. 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, e.g., in different states, and may be checked multiple times during the diagnostic procedure. Also, the tests are run on state maintained at the Monitor without involving the PE, thus satisfying the design goal of non-intrusiveness. Model based testing discusses several methods of forming these rules automatically. Authors in [42], [43] discuss mechanisms for automatic rule derivations from formal models like 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 pre- and post-conditions in the rule events over a logical window of $\pm\Delta t$, the phase, measured from the logical

time of l_i. This is attributed to the assumption of jitter bound on the communication link, namely, that a message at the Monitor cannot arrive out of order with respect to another message more than Δt away, originated at the same PE. Each rule in 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 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 cover all possible misbehavior in the PEs. Also rules that depend on 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 amongst the Monitors comes into play when a suspect node identified by an LM lies outside its domain, i.e. 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 local Monitor LM_i verifying it.
2. LM_i constructs the suspicion set SS_N for the failure and adds it to SS_{FN}.
3. For every $N' \in SS_N$ that belongs to the domain of LM_1 , LM_1 *tests* N' for correctness for the suspect link L' using rules from the SRB for that particular event and state. If N' is not faulty, then it is removed from SS_N and $SS_{N'}$ is added to the SS_{FN} queue.
4. For every N'' belonging to SS_N that is not under the domain of LM_i but 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 face of arbitrary failures of the Monitors. We assume Monitors are susceptible to *runtime* failures (e.g., 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 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 there can be failures on up to f LMs, each PE is verified by $2f+1$ LMs, called the Collaborative LM Set (denoted 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 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 CS_{LM} verify the same PE, they are spatially disjoint leading to possibly different views of the state of the PE.

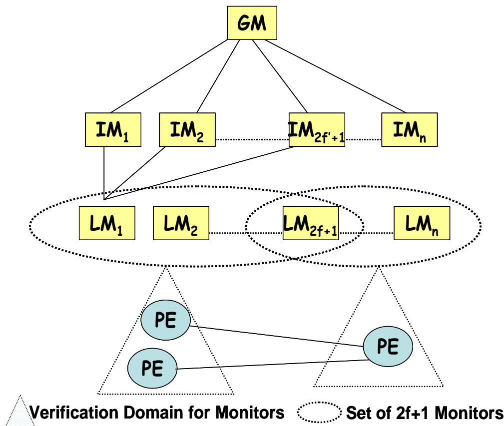


Figure 3: Redundancy in the Monitor hierarchy

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

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* using an existing method called the architectural-hybrid fault model [14] (Section 3.3.1).

The algorithm used by the LMs in a 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 CS_{LM} (including itself). When the atomic multicast delivers an LM the $(f+1)^{th}$ copy of the same alarm sent by different LMs in 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 Trusted Timely Computing Base (TTCB). The notion of architectural-hybrid fault model is simple: we assume different fault models for different parts of the system. Specifically, we assume 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 Byzantine resilient consensus, to a collection of external entities, like the LMs. These entities communicate in a world full of threats, 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.

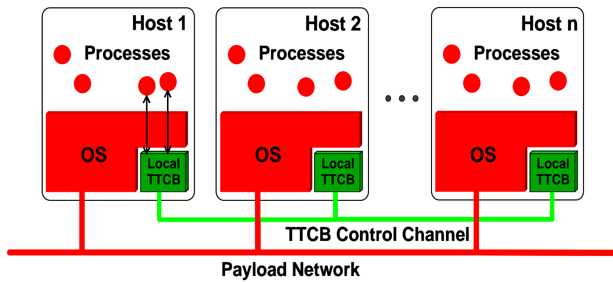


Figure 4: Architecture of *n* hosts with a TTCB

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 (Figure 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 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. While 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 we use is one of the simple services provided by the TTCB, 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*, e.g., 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, i.e., outside the TTCB. However, these channels guarantee the authenticity and integrity of the messages. These channels could be implemented using SSL or TLS. Finally, when the TTCB has information that *f* LMs received M, it gives M & all LMs in 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 (CS_{LM}) send alarms to all IMs in a Collaborative IM Set, denoted by CS_{IM}. 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, based on formal verification and static code analysis, can be used to build a fault-free GM. Possibility of faults in Monitors, forces an LM in CS_{LM} to accept a test request only if it receives *f+1* identical test requests from Monitors in 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 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 CS_{IM} initiates the diagnosis.

1. Failure F at PE N is detected by the CS_{LM} verifying it, which constructs the suspicion set SS_N and adds it to SS_{FN}.
2. The LMs assign an order to the alarm using the atomic multicast protocol and send an alarm along with SS_{FN} up to all the IMs in CS_{IM}.
3. The IMs wait for $f+1$ identical alarms and then start the diagnostic procedure.
4. For every $N' \in SS_{FN}$, the (correct) IMs in a CS_{IM} send test request to the CS_{LM} to verify N'.
5. Each $LM \in CS_{LM}$ that receives $f+1$ identical test requests from IMs in CS_{IM} *tests* N' for correctness of the suspect link L' using SRB.
6. The test results are sent above to the IMs in 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 SS_N and $SS_{N'}$ is added to SS_{FN}.
7. If a PE N'' lies outside the verification domain of the IMs in CS_{IM} then a *test request* 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 Local Monitors 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 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 (i.e., probability of detecting a faulty node is c_i) and a coverage d_i if the node has an error which 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 fault and propagated error over all nodes. Let the number of tests from SRB performed on the node be T and the total number of nodes be N. Each test yields an output O $\in$ {0, 1}, where an output 0 means the node passes the test and 1 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:

A = Prob(test=1|π) = c ; ***[1(a)]***

B = Prob(test=1| π′) = $d(1-(1-\rho\lambda)^k)$; ***[1(b)]***

Prob(*z-ones*| π) = $C(T,z)\, A^z\, (1-A)^{T-z}$ (where C is the binomial coefficient) ; ***[1(c)]***

Prob(*z-ones*| π′) = $C(T,z)\, B^z\, (1-B)^{T-z}$; ***[1(d)]***

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(π| *z-ones*) = *Prob*(*z-ones*| π).*Prob*(π) / *Prob*(*z-ones*) ; where *Prob*(z-ones) is given by *1(c)*.λ + *1(d)*.(1-λ) using the total probability formula.

$$= \frac{\lambda C(T,z) A^z (1-A)^{T-z}}{\lambda C(T,z) A^z (1-A)^{T-z} + (1-\lambda) C(T,z) B^z (1-B)^{T-z}} = \frac{1}{1+\frac{(1-\lambda)}{\lambda}\left(\frac{B}{A}\right)^z \left(\frac{1-B}{1-A}\right)^{T-z}} \quad ; \textbf{[1(e)]}$$

This equation 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 equation 1(b)

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

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

We claim that 1(e) is a monotonically increasing function of z. Note that A and 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 π=1) and one which is the victim of a propagated error (low probability of π=1). Therefore, A>B. Let us represent F(z) as $k\beta^z\mu^{T-z}$. For $A>B$, $\beta<1$ and $\mu>1$ and therefore Prob(π| *z-ones*) increases with z. This can also be proved through showing $d(Prob(\pi| \textit{z-ones}))/dz > 0$. This implies that the higher the value of z for a fixed T the greater is 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 asymptotically correct diagnosis for $N\rightarrow\infty$ for $k\geq2$ and $T\geq \alpha(N)log(N)$, where $\alpha(N)\rightarrow\infty$ arbitrarily slowly as $N\rightarrow\infty$. It is also optimal in diagnosis accuracy among diagnosis algorithms in its class.

Proof:

We observe that the posterior probability given by equation 1(e) matches the posterior probability of the FR algorithm [19]. Further it is proved in [19] that a diagnosis algorithm which produces this form of posterior probability is asymptotically optimal if T is chosen atleast as large as $\alpha(N)log(N)$ where $\alpha(N)$ grows arbitrarily slowly as compared to N. Note that tests T are can grow sub-linearly compared to the number of nodes to satisfy the condition. So if we

increase the number of nodes by Δ such that each additional node is similar in nature to any one of the existing N nodes, then tests from 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 (i) all testing is done with local knowledge, and (ii) a threshold number of tests needs to fail for an entity to be diagnosed as faulty. Hence the algorithm tends to perfect behavior asymptotically when $k \geq 2$ and T grows as $\alpha(N)log(N)$. Note that our diagnosis algorithm is also asymptotically correct for asymptotic behavior of T, independent of N since equation 1(e), $\lim_{T\to\infty}\lim_{z\to T} Prob(\pi \mid z-ones)$ approaches 1. Eqn. [1(e)] is an increasing function of z. Hence, we find the value $z = z_{th}$ which provides Prob(π| z-$ones$) = 0.5 and set the algorithm to conclude the node is faulty if $z > z_{th}$ and non-faulty otherwise. Equating eqn. 1(e) to 0.5 and simplifying we get:

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

Therefore, using the property of Prob(π| z-$ones$) being an increasing function of z and Theorem 1 in [39], we conclude that our diagnosis algorithm is optimal in its class 3AM. It is important to note that although the model considered in [19] and this paper are completely different, however

the equations for a posteriori probability have the same form, making our diagnosis algorithm optimal. ■

5 IMPLEMENTATION, EXPERIMENTS & 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 (RH), 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 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 5000 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 *head bind* messages sent by a receiver within 500ms 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 strict rule (as opposed to normal rule base rules used for detection) with *HI* denoting *hybrid incoming* rule. It signifies that the pre-condition event is incoming and the post condition event is outgoing. If in state S2, the receiver has received a data packet (E11) say with

linkID as *d* then there must be an *ack* packet (E9) within the phase interval around *d*. This rule ensures the receiver sends an *ack* packet on 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 linkID (logical time stamp) is assigned to the link along with the physical time, state, and event type. Link contains two IDs, one for the node which 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→B will be matched once the message sent by A and the corresponding one received by B is 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 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 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 & 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 on successfully receiving a data packet. Errors in message transmission can indeed be detected by checksum computed on the header. However, the Monitor is responsible for detecting & diagnosing errors in the protocol itself, which are clearly outside the purview of 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 kind of injections for a *burst length* period of time:*(a)Stuck-At injection*: 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 *(b) Directed Injection*: 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. *(c) Specific Injection*: Specific injections consist of specific procol errors like slow data rate, dropping acks, and hello message flooding. Burst error is

chosen as the fault model over single error since the protocol is robust enough that single errors are almost always tolerated by inbuilt mechanisms in the protocol.

5.3 Test Set Up and Topology

Figure 5(b) illustrates the topology used for the accuracy and the latency experiments on TRAM with components distributed over the campus network (henceforth called TRAM-D), while Figure 5(a) shows the topology for the local deployment of TRAM (TRAM-L). TRAM-D is important since a real deployment will likely have receivers distant from the sender. TRAM-L lets us control the environment and therefore run a more extensive set of tests (e.g., with a large range of data rates). The PEs and the LMs are capable of failing, while 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 min. data rate in TRAM needed to support the quality of the video application is set at 25 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 1Gbps links and each cluster machine is connected to a router through a 100Mbps link.

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.

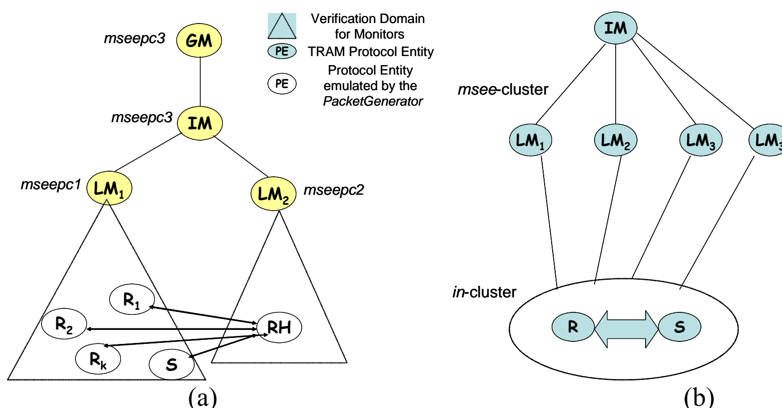


Figure 5: Topology used for accuracy and latency experiments in (a)TRAM-L (b)TRAM-D

Although Figure 5(a) 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. 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 incomplete termination of the algorithm. We perform experiments with both the optimistic and the pessimistic approach 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 (abbreviated as, *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 300ms for each injected fault. Inter packet delay is varied to achieve the desired increase in the data rate. Delay *d* is inserted using Gaussian random variable with mean *d* and standard deviation *0.01d*. Each point is averaged over 4 injections and 20-58 diagnosis instances, depending on the number of detections, which in turn depends on the rate of incoming faulty packets.

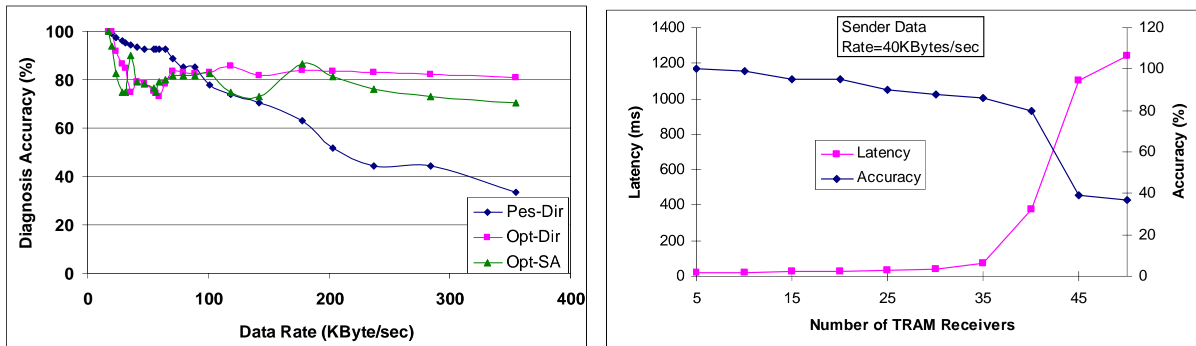


Figure 6: (a) Variation of Diagnosis Accuracy with Data Rate of Sender (b) Latency and Accuracy with increasing number of receivers.

Figure 6(a) shows that for *Pes-Dir* accuracy is a monotonically decreasing function with data rate. Diagnosis accuracy drops to a low of 33% for data rate at 355 KByte/sec. 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. Lack of adequate buffer causes packet drops leading to missing links in the causal graph leading to a drop in accuracy. Another factor is 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 small data rate but decreases with the increase in data rate. Unlike *Pes-Dir*, here the accuracy does not drop below 80%. 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 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 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.

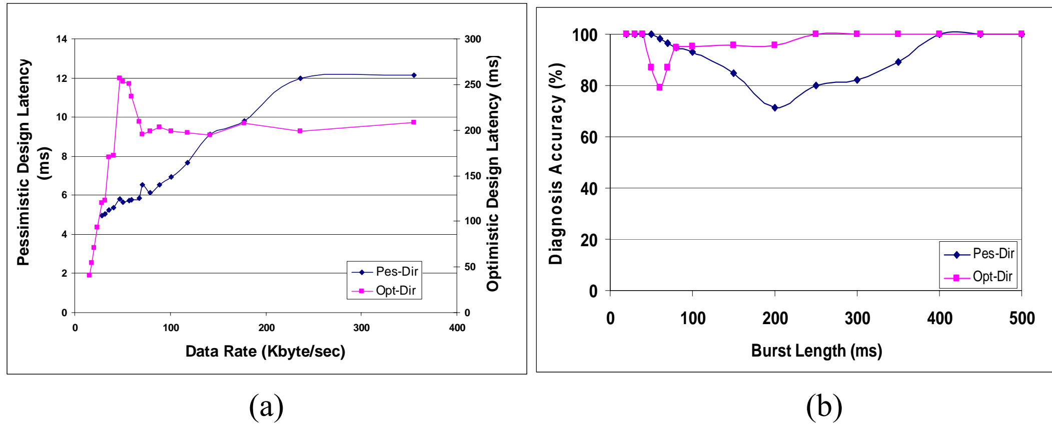
Figure 6((b) shows the scalability results where latency and accuracy of diagnosis is observed over increasing number of receivers. We employ the TRAM-L topology depicted in Figure 5(a) with each receiver receiving data at 40 Kbytes/sec. Thus, the total incoming rate for each receiver at the Monitor 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 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 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 causing diagnosis to fail. If an IM is overloaded, then it may drop incoming messages containing the SS from the LMs, leading to incomplete diagnosis.

Figure 7 (a) graphs the latency of diagnosis with 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 data rate which is expected because there are more packets to be tested by each rule in the SRB. Latency tends to saturate at high data rates because of incomplete causal graph leading to an inaccurate early termination. On the other hand in the *Opt-Dir* scenario, the latency keeps increasing with data rate because of the lazy link matching.

Effect of burst length: We study the impact of burst length on diagnosis accuracy for. We keep the data rate low at 15 KBytes/sec to isolate the effects due to high data rate. Diagnosis as shown in Figure 7 is accurate for low and high values of burst length. For small burst length, a small number of incorrect packets gets injected leading to a 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 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.

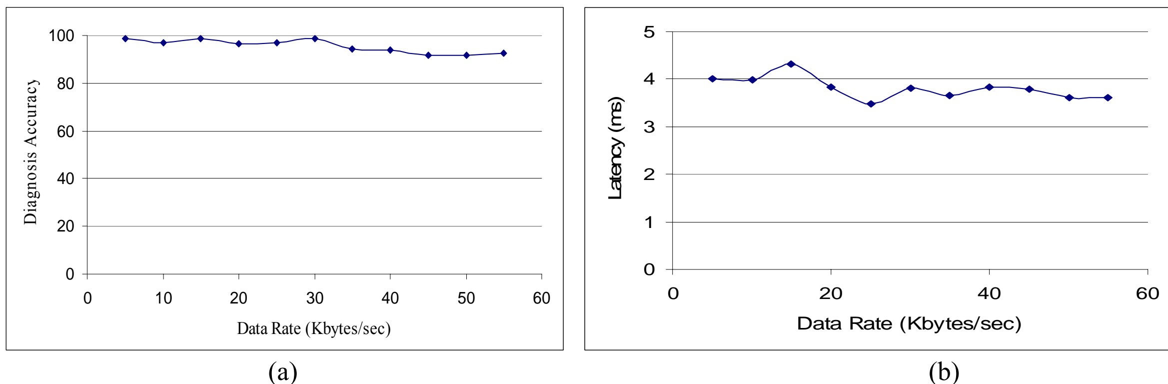


(a) (b)

Figure 7: (a) Variation of Latency with Data Rate and (b) Diagnosis Accuracy with Burst Length for Optimistic and Pessimistic Approaches

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. Figure 8(a) shows that the accuracy of diagnosis drops from a high of 98% at 15 KB/s to 91% for 50 KB/s. 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.



(a) (b)

Figure 8: (a) Diagnosis Accuracy and (b) Latency variation with increasing data rate of sender in TRAM-D

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 accuracy. Figure 8(b) shows the latency of diagnosis with increasing data rate. Intuitively when the data rate increases, increasing load on the Monitor should cause the latency to increase. However, the data rate used is low enough that it has no significant effect.

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 heart beats, watchdogs, and Intrusion Detection Systems (IDS). 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 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 ([25], [26]). King *et al.* proposed building dependency graph using logs at 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]. Authors provide a dyanamic data management system which is used to maintain aggregate 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 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 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 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 the authors in [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 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 black-box applications and has similarities to the Monitor approach ([29][31][32]). For example, in [29], the debugging system performs analysis of message traces to determine the causes of long latencies. However, in all of this work, the goal is not diagnosis of faults, but 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. 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 distributed diagnosis of failures in the protocol entities 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% 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.

References

[1] META Grp, "Quantifying Performance Loss: IT Performance Engineering and Measurement Strategies," 2000.
[2] FIND/SVP, "Costs of Computer Downtime to American Businesses," 1993.
[3] G. Khanna, J. Rogers, and S. Bagchi, “Failure Handling in a Reliable Multicast Protocol for Improving Buffer Utilization and Accommodating Heterogeneous Receivers,” In the 10th IEEE Pacific Rim Dependable Computing Conference (PRDC), pp. 15-24, 2004.
[4] G. Khanna, P. Varadharajan, and S. Bagchi, “Self Checking Network Protocols: A Monitor Based Approach,” In the 23rd IEEE Symp. on Reliable Distributed Systems (SRDS), pp.18-30, 2004.
[5] M. Diaz, G. Juanole, and J.-P. Courtiat, “Observer-A Concept for Formal On-Line Validation of Distributed Systems,” In IEEE Trans. on Software Engineering, 20(12), pp. 900-913, 1994.
[6] M. Zulkernine and R. E. Seviora, “A Compositional Approach to Monitoring Distributed Systems,” In IEEE Dependable Systems and Networks (DSN), 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,” In the Intl. Conf. on Dependable Systems and Networks (DSN), pp. 225-234, 2001.
[8] R. Buskens and R. Bianchini Jr., “Distributed On-line Diagnosis in the Presence of Arbitrary Faults,” In 23rd Intl. Symp. on Fault-Tolerant Computing (FTCS), 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,” In INFOCOM ’02, pp.1209-1217, 2002.

[10] T. Chandra and S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” In Journal of the ACM, 43(2), pp. 225–267, 1996.
[11] G. Bracha and S. Toueg , “Asynchronous Consensus and Broadcast Protocols,” In J. of the ACM, 32(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,” In the 23rd Intl. Symp. of Reliable and Distributed Systems (SRDS), pp. 174–183, 2004.
[13] M. Correia N. F. Neves, and P. Veríssimo, “The Design of a COTS Real-Time Distributed Security Kernel,” In the 4th European Dependable Computing Conference (EDCC), pp. 234–252, 2002.
[14] M. Correia and N.F. Neves and L. C. Lung and P. Veríssimo, “Low Complexity Byzantine-Resilient Consensus,” In Distributed Computing, 17(3), pp. 237--249, 2005
[15] A. Mostefaoui, M. Raynal, and C. Travers, “Crash-Resilient Time-Free Eventual Leadership,” In the 23rd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS), pp. 208-217, 2004.
[16] I. Katzela and M. Schwartz, “Schemes for Fault Identification in Communication Networks,” In IEEE/ACM Trans. On Networking, 3(6), pp. 753-764, 1995.
[17] F.P. Preparata, G. Metze and R.T. Chien. “On the Connection Assignment Problem of Diagnosable Systems,” In IEEE Trans. on Electronic Computers, Vol. EC-16, No. 6, pp. 848-854, 1967.
[18] S. Maheshwari and S. Hakimi, “On Models for Diagnosable Systems and Probabilistic Fault Diagnosis,” In IEEE Trans. on Computers, C-25, pp. 228–236, 1976.
[19] D. Fussel and S. Rangarajan, "Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity," In the 19th Intl. IEEE Symp. on Fault-Tolerant Computing (FTCS), pp. 560-565, 1989.
[20] M. Barborak , A. Dahbura , and M. Malek, “The consensus problem in fault-tolerant computing,” ACM Computing Surveys (CSUR), v.25 n.2, p.171-220, June 1993
[21] A. Bagchi and S. Hakimi, “An Optimal Algorithm for Distributed System Level Diagnosis,” In the 21st Intl. Symp. on Fault Tolerant Computing (FTCS), pp. 214-221, 1991.
[22] R. Chillarege and R. K. Iyer, “Measurement-Based Analysis of Error Latency,” In IEEE Trans. on Computers, 36(5), 1987.
[23] S. Lee and K.G. Shin, “On Probabilistic Diagnosis of Multiprocessor Systems Using Multiple Syndromes,” In IEEE Trans. Parallel and Distributed Systems, 5(6), pp. 630 – 638, 1994.
[24] A. Avizienis and J.-C. Laprie, “Dependable Computing: From Concepts to Design Diversity” In Proc. of the IEEE, 74(5), pp. 629–638, 1986.
[25] S. Chandra and P. M. Chen, “How Fail-Stop are Faulty Programs?” In the 28th Annual Intl. Symp. on Fault-Tolerant Computing (FTCS), pp. 240-249, 1998.
[26] H. Madeira and J. G. Silva, “Experimental Evaluation of the Fail-Silent Behavior in Computers Without Error Masking,” In the 24th Intl. Symp. on Fault-Tolerant Computing (FTCS), 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,” In Intl. Symp. on Integrated Network Management (IM), 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,” In the 12th Intl. Workshop on Distributed Systems: Operations & Management (DSOM), 2001.
[29] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, “Performance Debugging for Distributed Systems of Black Boxes,” In the ACM Symp. on Operating Systems Principles (SOSP), 2003.
[30] E. P. Duarte, and T. Nanya, “A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm,” IEEE Trans. on Computers, 47(1), pp. 34-45, 1998.
[31] J. L. Hellerstein, "A General-Purpose Algorithm for Quantitative Diagnosis of Performance Problems," Journal of Network and Systems Management, 2003.
[32] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, “Magpie: On-Line Modelling and Performance Aware Systems,” ACM-HotOS-IX, 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,” In Proc. of CAV 97: Computer-Aided Verification, LNCS, pages 340—351, 1997.
[34] K. Ravi and F. Somenzi. “High–Density Reachability Analysis,” In Proc. IEEE/ACM ICCAD’95, pages 154–158, San Jose, California, November 1995.
[35] J.R. Burch, E.M. Clarke, and D.E. Long, “Symbolic model checking with partitioned transition relations,” in DAC 91: Design Automation Conference, 1991, pp. 403–407.
[36] K.L. McMillan, Symbolic Model Checking: An Approach to the State-Explosion Problem. Kluwer Academic Publishers, Dordrecht, 1993.

[37] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, July 1978, 21(7):558-565.
[38] M. Castro and B. Liskov "Proactive Recovery in a Byzantine-Fault-Tolerant System," Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI '00), October 2000.
[39] S. Lee and K. Shin, "Optimal and efficient probabilistic distributed diagnosis schemes," IEEE Transacions on Computers, vol. 42, no. 7, pp. 882-886, 1993.
[40] S. T. King, P. M. Chen, "Backtracking Intrusions", *Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP)*, October 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 Transactions on Computer Systems (TOCS), Volume 21, Issue 2, pp. 164-206, 2003.
[42] J. Offutt and A. Abdurazik, " Generating Tests from UML Specifications," UML'1999, pp 416-429.
[43] C. Meudec, "Automatic Generation of Software Tests from Formal Specifications," PhD Thesis, The Queen's University 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, 34(3):375–408, September 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, pp. 55-96, Addison Wesley, 1993.