PROBABILISTIC DIAGNOSIS THROUGH NON-INTRUSIVE MONITORING IN DISTRIBUTED APPLICATIONS

GUNJAN KHANNA
MIKE YU CHENG
JAGADEESH DYABERI
SAURABH BAGCHI
MIGUEL P. CORREIA
PAULO VÉRISSIMO

TR-ECE – 05-19
DECEMBER 2005

PURDUE UNIVERSITY

SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, IN 47907-2035

Probabilistic Diagnosis through Non-Intrusive Monitoring in Distributed Applications

Gunjan Khanna, Mike Yu Cheng, Jagadeesh Dyaberi, Saurabh Bagchi
Dependable Computing Systems Lab
School of Electrical and Computer Engineering, Purdue University.
Email: {gkhanna,mikecheng,jdyaberi,sbagchi}@purdue.edu

Miguel P. Correia, Paulo Véríssimo
Faculty of Sciences
University of Lisbon, Portugal
Email: {mpc,pjv}@di.fc.ul.pt

Abstract

With dependability outages in distributed critical infrastructures, it is often not enough to detect a failure, but it is also required to diagnose the failure, i.e., to identify the source of the failure. Diagnosis is challenging because fast error propagation may occur in high throughput distributed applications. The diagnosis often needs to be probabilistic in nature due to imperfect observability of the payload system, inability to do white-box testing, constraints on the amount of state that can be maintained at the diagnostic process, and imperfect tests used to verify the system. In this paper, we extend an existing Monitor architecture, for probabilistic diagnosis of 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 & aggregate graph between the PEs based on their communication and uses this together with a rule base for diagnosing the failure. The Monitor computes for each suspected PE, a probability for the error having originated in that PE and propagated to the failure detection site. The framework is applied to a test-bed consisting of a reliable multicast protocol executing on the Purdue campus-wide network. Error injection experiments are performed to evaluate the accuracy and the performance overhead of the diagnostic process.

Keywords: Distributed system diagnosis, runtime monitoring, reliable multicast protocol, probabilistic diagnosis, error injection based evaluation.

1 Introduction

The connected society of today has come to rely heavily on distributed computer infrastructure, be it an ATM machine network, or the nationwide network of computer systems of the different power supply companies and the regulatory agencies. The infrastructure, however, is increasingly facing the challenge of dependability outages resulting from both accidental failures and malicious failures. The consequences of downtime of distributed systems may be catastrophic. They range from customer dissatisfaction to financial losses to loss of human lives[3]. Financial cost at the higher end of the spectrum is $6.4M per hour of downtime for brokerage firms [33].

In distributed systems, the fault in one component can manifest itself as an error and go undetected for arbitrary lengths of time. This may cause the occurrence of error propagation whereby other components, which may be fault-free themselves, are “infected” by the error. The error finally manifests itself as a failure and is detected at a component distant from the originally faulty component. The uptime of a system, as measured by the availability, is usually quantified as the Mean time to failure/(Mean time to failure + Mean time to recovery). There is an enormous effort in the fault tolerance community to increase the reliability of the components in a distributed system, thus increasing the mean time to failure. There is also a growing number of efforts aimed at reducing the mean time to recovery [34]. An important ingredient in the mix for high availability in distributed systems is knowing which components to recover. There is thus the need of tracing back through the chain of errors propagated in the system to determine the component that originated the failure. This serves as the goal for the diagnostic system.

We take the approach of structuring the overall system into an observer or monitor system, which provides detection and diagnosis, and an observed or payload system, which comprises of the protocol entities (PEs). The Monitor system is able to observe the interactions between the PEs through external message exchanges though the PE internals are a black box. The detection of a failure, also done by the Monitor (and described in our previous work [2]) triggers the diagnosis process. For the diagnosis, the Monitor creates a *Causal Graph* (**CG**) denoting the causal relation between the PEs as evidenced through the message sends and receives. For a given failure, the Monitor creates a *Suspicion Set* (**SS**) of PEs that are suspected to have originated the failure. The intuition is that an observed message exchange may be the vehicle of error propagation. The Monitor proceeds to test the PEs in the SS systematically and ultimately presents the result of the diagnosis process through identifying a set of (one or more) PEs as faulty.

The Monitor architecture is applicable to a large class of message passing based distributed applications, and it is the specification of the rule base that makes the Monitor specialized for an application. It also operates asynchronously to the payload system and is hierarchical [35]. It is assumed that the Monitors can fail and their failures are masked through replication. To reduce the number of replicas needed for achieving consensus, the implementation of the Monitor system is based on a hybrid failure model by embedding a trusted distributed component, called the Trusted Timely Computing Base (TTCB) [7], in a subset of the Monitors.

The existing view of Monitor diagnosis is a deterministic process whereby the PE responsible for initiating the chain of errors can be deterministically identified. This is however, an over-simplification of reality. In practical deployments, it is often the case that the Monitor does not have perfect observability of the PE because the network between the Monitor and the PE is congested or intermittently connected. This is particularly feasible because the application is distributed with components spread out among possibly distant hosts, and the Monitor and the payload systems may be owned by different providers and run on different networks. Next, the Monitor itself has finite resources and may drop some message interactions from consideration due to exhaustion of its resources (e.g., buffers) during periods of peak load. It is desirable that the Monitor be non-intrusive to the payload system and therefore the testing process comprises testing invariants on properties of the payload system behavior deduced through the observation process. Thus, no additional test request is generated for the PE. However these tests are not perfect and may generate both missed and false alarms. Hence, a probabilistic model is needed to assess the reliabilities of the PEs. Finally, the nodes have different error masking abilities and thus different abilities to stop the cascade of error propagation. This masking ability is not known deterministically. All these factors necessitate the design of a probabilistic diagnosis protocol.

The goal of the probabilistic diagnosis process is to produce a vector of values called *Path Probability of Error Propagation* (**PPEP**). For the diagnosis executed due to a failure at node *n*, PPEP of a node *i* is the conditional probability that node *i* is the faulty PE that originated the cascaded chain of errors given the failure at node *n*. The PEs with the *k*-highest PPEP values or with PPEP values above a threshold may be chosen as the faulty entities.

Our approach to probabilistic diagnosis builds on the structures of Causal Graph and Suspicion Tree from the deterministic diagnosis protocol. A probabilistic model is now built for each of node reliability, error masking ability, link reliability, and Monitor overload. The probability values for some of the components are partially derived from history maintained at the Monitor in a structure called the *Aggregate Graph* (**AG**). A consequence of moving the fine-grained information from the CG to the summarized AG is that it reduces the amount of state to be maintained at the Monitor. The probability values from each component are combined for the nodes and the links in the path from a node *i* to the root of the Suspicion Tree to come up with *PPEP(i)*. The combination has to be done with care since the probabilities are not all independent. For example, overload condition at the Monitor is likely to persist across two consecutive messages in the payload system.

The Monitor system is demonstrated by applying it to a distance e-learning application used at Purdue, which uses a tree-based reliable multicast protocol called TRAM[4]. The TRAM components and distributed Monitor components (a two level hierarchy is used) are installed on hosts spread over Purdue's campus. Different kinds of errors, both low and high level, are injected into the message headers. The results show the capabilities of the Monitor system (diagnosis coverage and latency) as a function of the message rate and the resources at the

Monitor. It is found that unlike in the deterministic case, the size of the buffer to be searched for the faulty entity has a profound effect on the latency and the accuracy of diagnosis.

The contributions of the paper can be summarized as follows:

1. The paper provides the design of a scalable non-intrusive diagnosis infrastructure for distributed applications for diagnosing arbitrary failures. The diagnosis can be achieved in the face of Byzantine failures in the Monitor itself and error propagation across the entire payload system.
2. The paper presents a protocol for making the diagnosis accurate under realistic deployment conditions of imperfect tests, imperfect observability of the payload system, and finite resources at the Monitor. Feedback mechanisms built into the system enable progressive refinement of the numerous probability parameters, thereby reducing the requirement for frequent manual tuning of the system.
3. The design is realized through an actual implementation, which is demonstrated on a sizable third-party application. Detailed error injection brings out the capabilities and limitations of the system.

The rest of the paper is organized as follows. Section 2 presents the failure and system model and background from deterministic diagnosis. Section 3 presents the probabilistic diagnosis protocol. Section 4 presents the experimental testbed along with the configuration of the payload and the Monitor systems. Section 5 gives the experimental results. Section 6 reviews related work and Section 7 concludes the paper. A list of abbreviations used in the paper is provided for reference in Appendix A.

2 System Model and Background

2.1 System and Failure Model

The system consists of two parts – the observer and the observed. The observed component comprises of the application protocol entities (PEs) which are being monitored for failures. The observer consists of the hierarchical Monitor architecture responsible for performing *detection* and *diagnosis*. Each individual Monitor performs matching of messages against a set of rules specified as an input. The Monitor obtains the protocol either 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) forwards each message to the overseeing Monitor. In either scenario the internal state of the PEs is not visible to the Monitor and the PEs are treated as black-box for the diagnostic process. The *Diagnosis Engine* of the Monitor is triggered when a failure is detected.

The system comprises of multiple Monitors logically organized into Local, Intermediate, and Global Monitors. The *Local Monitors* (LMs) directly verify the PEs while an *Intermediate Monitor* (IM) collects information from several Local Monitors. An LM filters and sends only aggregate information to the IM. Since in scalable distributed applications, most of the interactions are local, it is expected that most messages will get filtered at the LMs and not be visible to the higher levels of the Monitor hierarchy, thereby making the system scalable.

We assume that PEs can fail arbitrarily, exhibiting Byzantine failures that are observable in external messages. We follow the classical definition of faults being underlying defects that are triggered to become errors and some errors causing end-user visible failures. Errors can propagate from one PE to another through the message exchanges between them. The Monitor detects failures in PEs by comparing the observed message exchanges against the *normal* rule base (*normal* to distinguish it from the *strict rule base* used during diagnosis).

The communication between the PEs could be asynchronous while jitter on any given link between a PE and the Monitor system is bounded. The Monitor maintains a logical clock for each verified PE and it is incremented for each event – a message send or receive. The assumption required by the diagnosis protocol is that for an *S-R* communication, 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 less than a constant Δt, called the *phase*. 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 $t_{L2} \geq t_{L1}+\Delta t$. Unreliable communication channel is considered where message duplication, loss or conversion to another correct protocol message may happen.

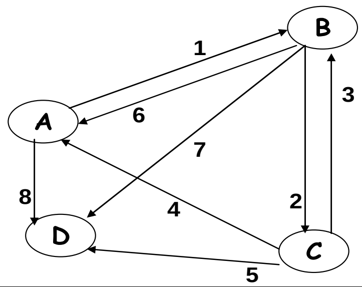
Monitors may have faults which can cause an arbitrary failure. We use replication amongst LMs and IMs to mask these failures. The Monitors are enforced to have a hybrid fault model through the TTCB module [6] which helps in reducing the number of replicas from $3f+1$ to $2f+1$. The regular communication channel between the Monitors is assumed to be asynchronous while the dedicated control channel used by the TTCB is synchronous.

2.2 Deterministic Diagnosis

In previous work [35], we have used the Monitor architecture to perform deterministic diagnosis. The probabilistic diagnosis model described in this paper is obtained by relaxing the assumptions made in the deterministic diagnosis. Here we explain the important components of the deterministic diagnosis which are required for understanding of probabilistic diagnosis.

2.2.1 Causal Graph

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 or link *e* contained in E, between vertices *v1* and *v2* (which represent PEs) indicates interaction between *v1* and *v2* and contains state information about all observed message exchanges between *v1* and *v2* including the logical clock (LC) at each end. The links are also time-stamped with the local (physical) time at the Monitor, at which the link is created. An example of a CG created at the Monitor is given in Figure 1 for the sequence of message exchange events shown with PEs A, B, C, and D. The number denotes the sequence of the message. For example for message '6' logical clock time at the sender is B.LC4. Since message '2' is assigned a logical time value of B.LC2 which causally precedes message '6'. The order of the messages is the order seen by the Monitor which may be different from the order in the application because the communication links are asynchronous.



Message ID	Sender.LogicalClock , Receiver.LogicalClock
1	A.LC1, B.LC1
8	A.LC4, D.LC3
2	B.LC2, C.LC1
6	B.LC4, A.LC3
7	B.LC5, D.LC2
3	C.LC2, B.LC3
4	C.LC3, A.LC2
5	C.LC4, D.LC1

Figure 1: A sample causal graph. A, B, C and D exchange messages 1-8 among each other. The message ID indicates the causal order.

2.2.2 Suspicion Set

Detection of failure, say at node *N* in the CG, starts the diagnostic procedure. Henceforth, we will use the expression "failure at node *N*" for a failure detected at the PE corresponding to the CG node *N*. Diagnosis starts at the failure node *N* where the rule is initially flagged, proceeding to other nodes being *suspected* for that failure. 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 recent past denoted by SS_N. If a failure is detected at node *N* then initially $SS_{FN}=\{SS_N\}$.

Each of the nodes in SS_{FN} is tested using the strict rule base (SRB). 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> <State1> <Event1> <Count1> <State2> <Event2> <Count2>

where, *Type* depends on whether the incoming, outgoing, or a hybrid link is being tested, (*State1, Event1, Count1*) forms the precondition to be matched, while (*State2, Event2, Count2*) forms the post-condition that should be satisfied for the node to be deemed not faulty. The examination of *Event 2* is done over a phase around *Event 1* in the precondition. SRB of form <S, E, C> refers to the fact that the event *E* should have been detected in the state *S* at least count *C* number of times. Information about the State and Events is stored by the Monitors and updated on viewing message exchanges. An example of a SRB rule is *HI S2 E11 1 S2 E9 1*. The hybrid rule states that if in state *S2*, the receiver has received a data packet (*E11)*, then there must be an *ack* packet sent out within the phase interval around the data packet. The rule for checking is determined by the combination of the state of the PE and the event.

The SRB yields a 1(0) if the node is faulty (fault-free) deterministically. If a node $n_i \in SS_{FN}$ is found to be fault-free then it is removed from the Suspicion Set. If none of the nodes is found to be faulty then in the next iteration, the Suspicion Set 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 $SS_{FN} = \{SS_{n_1}, SS_{n_2} \ldots, SS_{n_k}\}$. The procedure of contracting and expanding the Suspicion Set repeats recursively until the faulty node is identified or a *cycle boundary* is reached. *Cycle* is defined as a termination point for the diagnostic process. This may arise from bounds on the error latency, bounds on the error propagation due to repeating states and deterministic bugs which manifest deterministically in the same state, or recovery points at which any latent errors are flushed, e.g., through reboot [5].

3 Probabilistic Diagnosis Model

The model of the payload system assumed in the deterministic diagnosis process is overly simplistic in several deployments. The relative placement of the Monitor and the verified PEs may cause imperfect observability of the external messages. The Monitor may be resource constrained and may not be able to accommodate periodic surges in the rate of exchanged messages among the PEs. The tests used to diagnose the PEs may be imperfect and the inherent characteristics of the PEs, e.g., their error masking capabilities, may not be accurately known to the Monitor system. The probabilistic diagnosis protocol handles these limitations, which were assumed away in the deterministic protocol. For a given failure the goal of the probabilistic diagnosis process is to assign a probability measure for every node to be the cause of that failure.

It may be infeasible storage wise to keep the entire state of all PE interactions till the cycle boundary. Perhaps more importantly, performing diagnosis on all the nodes till the cycle boundary will make the latency of the diagnosis process unacceptably long. However, it is not desirable to completely flush the old state in the CG as the prior information could be utilized to provide historical information about a PE's behavior which may aid in the diagnostic process. Our solution is to aggregate the state information in the CG at specified time points and storing it in an *Aggregate Graph* (AG).

As in the deterministic diagnostic approach, incoming information in the Monitor is initially stored in a Temporary Links (TL) table where it is organized based on the source node, destination node and event type as the primary keys. For this link to be completed in the CG, 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. This information has to be matched and organized into the CG for diagnostic purposes. When the high water mark for the TL (HW_{TL}) is reached, then as many links as can be matched are transferred to the CG while those that cannot be matched but are within the phase from the latest message are kept in the TL. Remaining links in the TL are moved to the CG as unmatched links.

3.1 Aggregate Graph

The Aggregate Graph contains aggregate information about the protocol behavior averaged over the past. The AG is similar to CG in the structure i.e. a node represents a PE and a link represents a communication channel. The link is formed only if at least one message in the past has been exchanged between the entities. The links are directed and unlike the CG there is a single directed link between A and B for all the messages which are sent from A to B. Each node and link has some aggregated information stored and continuously updated which aids in the final diagnosis. The AG contains some node level information (such as, the node reliability) and some link level information (such as, the reliability of the link in the payload system). These information fields are formally defined in Section 3.2.

The time duration between consecutive CG to AG conversions is referred to as a *round*. State information is transferred from the CG to the AG if the high water mark (HW_{CG}) is reached after a TL to CG conversion. The amount of information kept in the CG is equivalent to that for a phase around the latest event. It is important for diagnosis accuracy that information stays for some time in the CG and is not immediately transferred to the AG. Therefore the size of the CG should be significantly higher than that of the TL.

3.2 Probabilistic Diagnosis

The operation of the diagnosis protocol has two logical phases: (1) The actual diagnostic process that results in a set of nodes being diagnosed as cause of failure; (2) Information from the diagnostic process being used to update the information present in the AG. Let us first look at the diagnostic process.

3.2.1 *Diagnosis Tree*

As in the deterministic diagnosis case, the CG is used to calculate the set of suspicion nodes, tracing back from the node where the failure was detected. A *Diagnosis Tree (DT)* is formed for failure *F* at node *D*, denoted as $DT_{FD.}$ The tree is rooted at node D and the nodes which have directly sent messages to node D denoted by SS_{D1} are at depth 1 and so on. Since the CG is finite size, the tree is terminated when no causally preceding message is available in the CG after some depth *k*.

The sample DT created from the sample CG in Figure 1 is shown in Figure 2. The numbers at the links correspond to the link IDs. The path *P* from any node *N* to the root *D* constitutes a possible path for error propagation and the probability of path *P* being the chain of error propagation is given by the *Path Probability of Error Propagation* (PPEP).

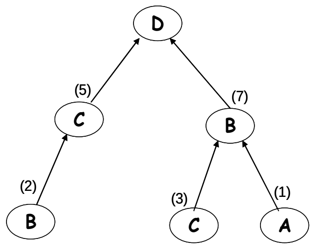


Figure 2: Sample DT for the CG in Figure 1.

Definition: *PPEP(N, D)* is defined as the probability of node *N* being faulty and causing this error to propagate on the path from *N* to *D*, leading to a failure at *D*. This metric depends on the following parameters:

(1) **Node reliability** – The node reliability is a quantitative measure of the PE corresponding to the node being faulty. The PPEP for a given node is proportional to its node reliability. The node reliability is obtained by running the tests from the SRB relevant to the current state and the event at the node in the CG. The result from the CG is aggregated with the previously computed node reliability (n_r) present in the AG. Let *c* be the combined coverage of the tests in the SRB. Then node reliability is updated as $n_r = (1 - \rho)c + \rho\ n_r$, where ρ is the weight used for current coverage. . The node reliability is maintained for each node in the AG.

(2) **Link reliability** – The link reliability quantifies the Monitor's estimate of the reliability of a link in the payload system. Since the Monitor does not have a separate probe for the quality of the link, it estimates link reliability (l_r) by the fraction of matches of a message reported from the head of the edge (sender) with that reported from the tail of the edge (receiver). The PPEP for a given node is proportional to the link reliability, because high link reliability increases the probability of the path being used for propagating the error. The link reliability is maintained for each edge and each event in the AG.

(3) **Error Masking Capability (EMC)** – The error masking capability (e_m) quantifies the ability of a node to mask an error and not propagate it through the subsequent links on the DT towards the root node D. The PPEP for a given node is inversely proportional to the EMC values of nodes in the path since the intermediate nodes are less likely to have propagated the error to *D.* With the DT in Figure 2,

$$PPEP(C, D) = n_r(\mathrm{C}) \cdot l_{r(C,D)}\,;\ PPEP(B, D) = n_r(\mathrm{B}) \cdot l_{r(B,C)} \cdot (1- e_m(C)) \cdot l_{r(C,D)}$$

Note that collusion among PEs reduces the coverage of the diagnosis when active forwarding is used. Thus, a sequence of PEs in a chain may omit to forward the messages to the Monitor though they are sent in the payload system. This will cause the Monitor to reduce the PPEP of the path through the colluding PEs. For an autonomous system, the parameters used in the diagnosis process should be automatically updated during the lifetime of the system as more failures and message interactions are observed and this forms the topic of our discussion next.

3.2.2 Calculating Node reliability

The objective is to assign node reliabilities to nodes corresponding to PEs, based on the results of the rules from the SRB. Let the set of tests that can be applied to the node *i* based on the event and the state be T_i. This set is partitioned into two sets A and A', depending respectively on if the test returned a value of 1 or 0. The weight of test $T_{i,j}$ is $w_{i,j}$. Then, the reliability of node *i* is given by $n(i) = \sum_{T_{i,j} \in A'} w_{i,j} \,/ \sum_{T_{i,j}, \forall j} w_{i,j}$.

The weight of a test is proportional to the following factors: the frequency of invocations ($w^{(f)}$) where the test gave the correct result, i.e., agreed with the ultimate diagnosis by the Monitor; and whether the test examines state for a period of time greater than the transients in the system ($w^{(r)}$). The overall weight is calculated as $w_{i,j} = w_{i,j}^{(f)} . w_{i,j}^{(r)}$, where the two terms correspond to the two factors.

3.2.3 Calculating Link Reliability

At the time of formation of the AG , link reliability of the edge from A to B is calculated as follows:

$l_{r(A,B)} = n_m / n_t + n_m$ where n_m = Number of matched edges for A to B communication and n_t = Number of unmatched edges from A to B.

Subsequently, when CG to AG conversion takes place, link reliability in AG ($l_{r(A,B)}$) is updated with the link reliability for the current round ($l_{r\,c}$) as $l_{r(A,B)} = (1-\rho)l_{r\,c} + \rho l_{r(A,B)}$. Note that there may be multiple links between A and B for different states in which they have communicated. This design is influenced by the intuition that faults are state dependent. Notice that in the PPEP calculation, the edge reliabilities of adjoining edges are multiplied though the events are actually not independent. The explanation is given considering a linear chain of communication from C to B and B to A. The probability of a successful communication from C to A is P(C→B is successful)•P(B→A is successful|C→B is successful). In the link reliability formulation, the dependence is implicitly taken into account since the matched and unmatched link count on the B→A link is affected by the events on the C→B link.

3.2.4 Calculating Error Masking Capability

Assume that in the DT of Figure 2, nodes C and A at the same depth 2 are both faulty but PPEP(C) is the highest and PPEP(A) is low whereby node C is diagnosed as faulty and node A is not. In the Monitor system, node B is taken to have masked the error and not propagated it if the following three conditions are satisfied: (i) Running SRB rules on B yields a low value (*c(B)*); (ii) Running SRB rules on A yields a high value (*c(A)*); and (iii) Link reliabilities $l_{r(A,B)}$ and $l_{r(B,D)}$ are high (to ensure that the error must have propagated). The increment Δ(*EMC*) is thus

$$\Delta(EMC) = EMC_{\text{prev}} \frac{c(A) \bullet l_{r(A,B)} \bullet l_{r(B,D)}}{c(B)} \text{ and } EMC_{\text{new}} = EMC_{\text{old}} + \Delta(EMC)$$

We decrease the EMC for every intermediate node residing on a path which is finally diagnosed to have caused the error propagation.

3.3 Adjusting the PPEP using P_{block}

Monitors could be deployed on heterogeneous hosts having different buffer sizes and processing capacities and they may verify entities with widely varying message rates. If the buffer in a Monitor is filled, any incoming message would be dropped without processing causing the link reliability to artifically go down. Therefore we calculate the blocking probability at the Monitor by modeling it as an *M/D/n/N* queue – exponential arrival rate λ, deterministic processing time μ, n number of threads simultaneously servicing the incoming packets, and the maximum buffer size N. Solving the queuing model, the probability of blocking is derived as:

$$P_{\text{block}} = X/Y \text{ ; where } X = \frac{a^N}{n!\mu^{N-n}} \text{ and } Y = \sum_{i=0}^{n} \frac{a^i}{i!} + \sum_{i=n+1}^{N} \frac{a^i}{n!\mu^{i-n}} \text{ ; where } a = \lambda/\mu$$

With arrival rate smaller than the processing time and infinite buffer size, the blocking probability tends to zero. The l_r calculated by the Monitor is adjusted to $max\{ l_r/(1\text{-}P_{\text{block}}), 1\}$.

3.4 Distributed PPEP

The PEs may be spanning several networks and even organizational boundaries and be verified by different Monitors each of which constructs a part of the AG and the CG obtained from its local information. During diagnosis it is quite likely that the DT contains PEs which are verified by some other Local Monitors. This entails the requirement of distributed diagnosis. Complete transfer of the local AGs and CGs to construct global information at higher level Monitors is not scalable. Instead, we leverage the fact that due to the multiplicative form of the PPEP computation, the PPEP value can be computed incrementally by each Monitor for the part of the DT under its verification domain. Assume in Figure 2, that nodes B and C are monitored by LM_1 and C and D by LM_2 and the diagnosis is performed by LM_2. In order to calculate PPEP(B, D) for the path (B-C-D) LM_2 needs n_r(B) , $l_{r(B,C)}$, e_m(C), and $l_{r(C,D)}$ of which the first two are not available locally. Therefore, LM_1 sends PPEP(B,C) for a failure at C to LM_2. In general, for a path A, X_1 X_2 ….$X_{n,}$,B, PPEP(B, A) can be recursively written as:

$$PPEP(B, A) = l_r(A,X1) \cdot e_m (X_1) \cdot PPEP(B, X1)$$

4 Implementation and Experimental Testbed

4.1 Application

We deploy the Monitor system across the Purdue campus-wide network to monitor a reliable multicast protocol (TRAM) [4]. TRAM is a tree based reliable multicast protocol consisting of a single sender, multiple repair heads (RH), and receivers. It provides the guarantee of an uninterrupted message stream from the sender to all the receivers in the face of node and link failures. 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 for liveness.

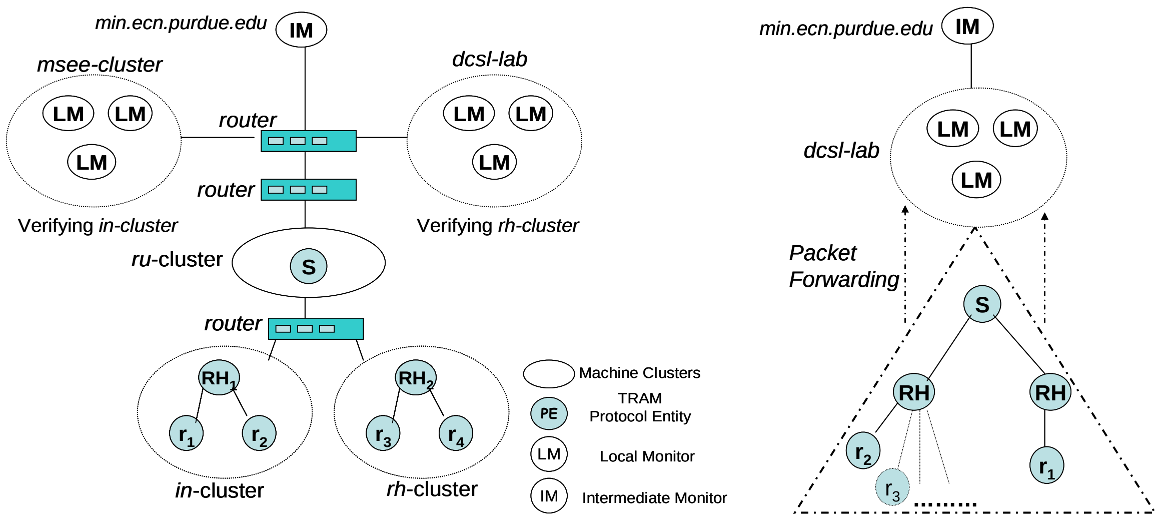


Figure 3: (a) Physical topology of test-bed (TRAM-D) (b) The emulated TRAM configuration (TRAM-L)

Figure 3(a) illustrates the topology used for the diagnosis experiments on TRAM with components distributed over the campus network (henceforth called TRAM-D), while Figure 3(b) shows the topology for an emulated local deployment of TRAM (henceforth called TRAM-L). 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). Running a larger configuration of receivers on TRAM-D was not possible because of a synchronization problem in the vanilla TRAM code obtained from [32] which causes receivers to disconnect unpredictably. The TRAM entities are verified by replicated Local Monitors with one Intermediate Monitor above. The TRAM entities do active forwarding of the messages to the respective LMs. The routers are interconnected through 1Gbps links and each cluster to a router through a 100Mbps link. Each cluster machine is Pentium III 930.33 MHz with 256 MB of RAM.

4.2 Rule Base

The Normal Rule Base (NRB) and the Strict Rule Base (SRB) for TRAM are input to the Monitor. The exhaustive enumeration of rules in the rulebases for the experiments are in the [36]. Recollect that the SRB verifies the messages sent by the PEs over the *phase* interval which is a much smaller window compared to that of NRB. A few examples of the SRB rules used for our experiments are: "*O S1 E11 1 S3 E11 30 1*" This rule states that if there is one data message (E11) in state S1, then at least 30 more E11 links should be present in the CG. The last value "1" is the weight assigned to this rule. For these experiments all SRB rules are assigned equal weights. "*HO S6 E1 1 S6 E9 1 1*" This hybrid outgoing rule (HO) verifies that on receiving a *hello* message(E1) in state S6, the receiving entity must send a *hello-reply* (E9) within the same phase for liveness.

4.3 Fault Injection

We perform *fault injection* in the header of the TRAM packet to induce failures. We choose the header since the Monitor's current implementation only examines the header. A PE to inject is chosen (TRAM sender or receiver) and a burst length worth of faults is injected. The fault is *injected* by changing bits in the header for both incoming and outgoing messages. A burst length is chosen since TRAM is robust to isolated faults. The burst may cause multiple detections and consequently multiple concurrent diagnoses. Note that the emulated errors are not simply message errors, but also symptomatic of protocol faults in the PEs. Errors in message transmission can indeed be detected by checksum on the header but protocol errors cannot. Following fault injection are used:

(a) *Random Injection*: A header field is chosen randomly and is changed to a random value, valid or invalid *w.r.t.* the protocol. If the injected value is not valid, then a robust application check may drop the packet without processing it.

(b) *Directed Injection*: A randomly chosen header field is changed to another randomly chosen value, which is valid for the protocol.

(c) *Specific injection*: This injection is carefully crafted and emulates rogue or selfish behavior at some PE.

5 Experiments and Results

Accuracy of diagnosis is defined as the ratio of number of correct diagnosis to the total number of diagnosis performed. Correct diagnosis is when the PE flagged as faulty, i.e. the PE with highest PPEP, is the PE that was injected with faults. The latency is the time measured from the point to detection to the end of diagnosis.

5.1 Latency and Accuracy for TRAM-D

5.1.1 Random Injection at Sender

Effect of Buffer Size: The fault injection causes error propagation to the RH and the receivers causing independent diagnoses at each entity. Figure 4 shows the latency and accuracy of diagnosis with increasing maximum buffer size for the CG. TRAM sender's data rate is kept at a low value of 15 Kbits/sec to avoid any congestion effect. Each data point is averaged over 300 diagnosis instances. Latency of diagnosis increases with buffer size since on an average the CG will be storing more links, leading to more nodes in the DT and hence higher processing for calculating the PPEP. The fundamental factor that determines the latency is the size of the DT, which depends on how full the CG was when the diagnosis was triggered, which is bounded by the CG buffer size. Diagnosis has a low accuracy for low CG buffer sizes as it is likely that the link connecting the faulty PE is purged from the CG during the CG-AG conversion. Higher CG size increases the accuracy because there are more links in the CG which increases the probability of SRB rules detecting errors leading to a high value of PPEP. Accuracy decreases with very high CG because several diagnoses do not complete as the size of the DT is large leading to an unacceptably high load on the Monitor.

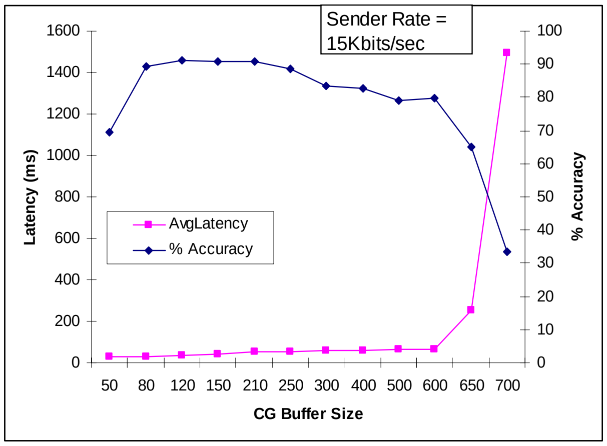
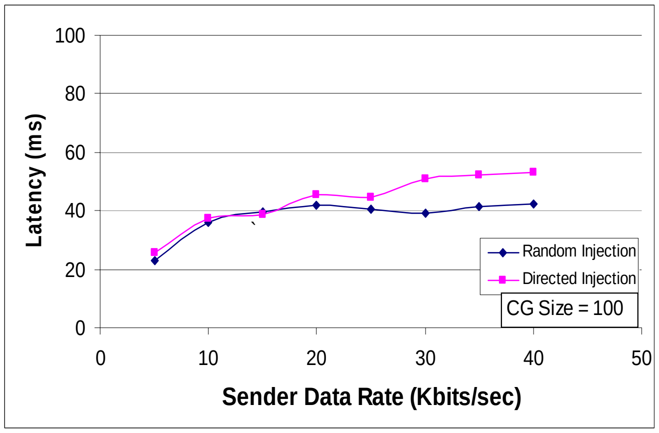


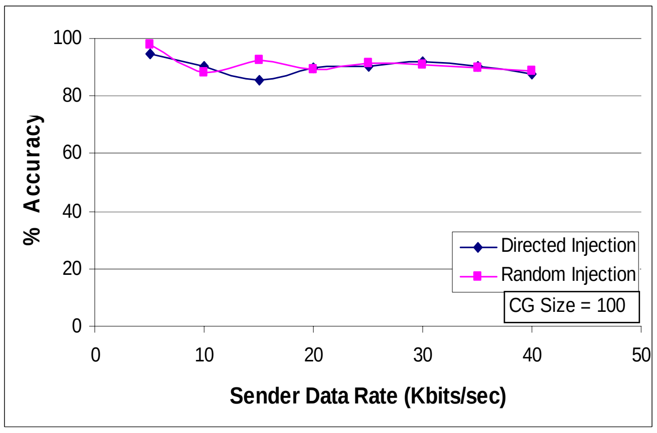
Figure 4: Latency and Accuracy for TRAM-D with random injection at sender

The increase in load with increasing CG buffer size is a direct consequence of the probabilistic diagnosis and was not present in the deterministic diagnosis. In probabilistic diagnosis, the entire CG is explored for faulty entities while with deterministic diagnosis, as soon as an entity is (deterministically) flagged as faulty, the process is halted. It is to be noted that during the diagnosis process, the TRAM entities are still sending packets to the Monitors leading to an additional detection overhead at the Monitor.

Effect of data rate: In this experiment, the buffer size at CG is fixed at 100 links, the data rate from the sender is varied and random injection is performed at the sender. Figure 5(a) shows the latency with increasing data rate. As the sender's data rate increases, the incoming packet rate at the Monitors increases by a multiplicative factor since each LM is verifying multiple PEs. Theoretically, till the Monitor's capacity is overrun, the sender data rate is expected to have no effect on latency since the CG buffer size is fixed and therefore the size of the DT being explored is fixed. It is not possible to see the breaking point (i.e. the "*knee*") in TRAM-D with the sender data rate since TRAM is robust and throttles the sending rate when the network starts to get congested.



(a)



(b)

Figure 5: Latency and Accuracy with increasing data rate for random and directed injection

5.1.2 Directed Injection at Sender

We repeat the above experiments with directed injection at the sender. From Figure 5(a), we discern that the latency is higher compared to random injection for the same data rate. This is because directed injection causes more valid but faulty packets. This leads to a higher number of state transitions in the Monitor's STD causing more diagnosis procedures to run resulting in an increased load on the Monitor. With random injection on the contrary, a larger fraction of packets is discarded by the Monitor and therefore resulting in a lighter load. The diagnosis accuracy follows a similar trend as the random injection scenario.

5.2 Latency and Accuracy for TRAM-L

5.2.1 Fault Injection at Sender

We emulate the topology depicted in Figure 3(b) on a local network to investigate the performance of Monitor in high data rate scenarios and use more receivers under an RH. First we fix the CG size at 100 links and vary the incoming data rate from 200 Kbits/sec to 1.9 Mbits/sec. Figure 6 depicts the latency and the accuracy variations with increasing data rate.

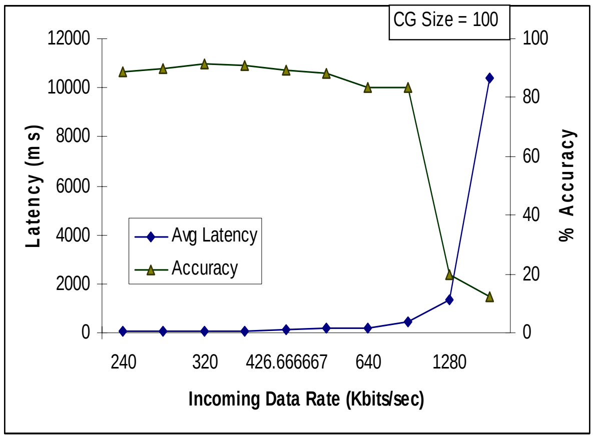


Figure 6: Latency and Accuracy VS data rate for directed injection at sender in TRAM-L

For low data rates the latency is about 300 ms and remains almost constant till a data rate of 1 Mbits/sec. Further increase in data rate causes the latency to rise exponentially because the Monitor's servicing rate is not able to keep up with the large CG size and incoming packet rate. We can see that this "knee" occurs at a lower CG size compared to TRAM-D (Figure 4) because of higher data rate. Accuracy is near constant for data rates up to 1 Mbits/sec and breaks beyond that because of incomplete diagnoses attributed to higher load on the system.

5.2.2 Fault Injection at Receiver

Fault injection is performed at the receiver to validate if the results are generalizable across PEs. The results of directed injection to the sender and a receiver are plotted in Figure 7. The plots are found to be almost overlapping thus indicating the closeness of the match.

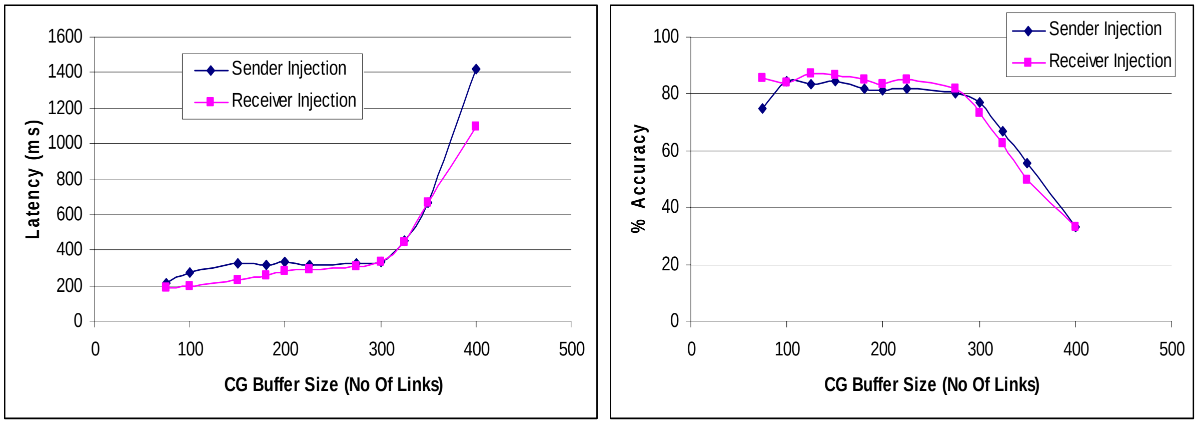


Figure 7: Latency and Accuracy for sender and receiver injections.

5.3 Specific Injection

We perform specific injection in TRAM-D to observe the effect of a rogue receiver and to precipitate error propagation to varying degrees. Receiver R_4 is modified to send *ack* at a slower rate (similar to [1]). Since in TRAM a cumulative *ack* is sent up the tree, R_4's misbehavior prevents RH_2 from sending the *ack*. This forces the sender to reduce the data rate because the previous buffer cannot be purged causing a slow data rate across.

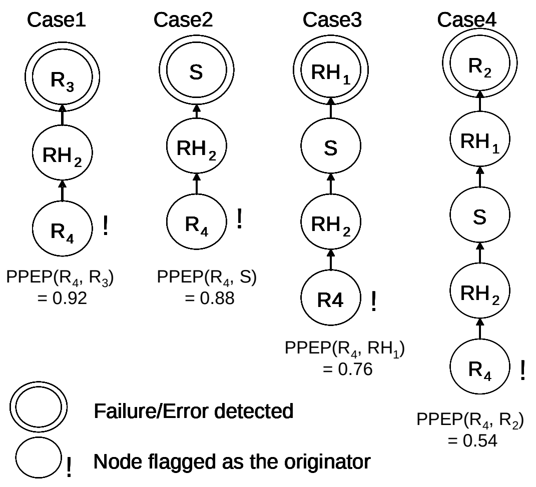


Figure 8: Parts of DTs formed during specific injection scenario in TRAM-D

Thus error propagation occurs across the entire protocol system. The detection engine reports detection at several PEs as shown in Figure 8. The diagnosis algorithm is able to diagnose the correct node (R_4) as faulty in all the cases since its PPEP value was highest in each DT.

6 Related Work

White box systems: The problem of diagnosis in distributed systems can be classified according to the nature of the payload system being monitored – white box where the system is observable and, optionally, controllable; and black box where the system is neither. White box diagnostic systems often use event correlation where every managed device is instrumented to emit an alarm when its state changes [15]-[17]. By correlating the received alarms, a centralized manager is able to diagnose the problem. Obviously, this depends on access to the internals of the application components. Also it raises the concern whether a failing component's embedded detector can generate the alert. This model does not fit our problem description since the target system for the Monitor comprises of COTS components, which *have to be* treated as black-box. A number of white box diagnostic systems that correlate alarms have been proposed in the intrusion detection area [11][12]. An alternative diagnostic approach is to use end-to-end probing [18]-[20]. A probe is a test transaction whose outcome depends on some of the system's components; diagnosis is performed by appropriately selecting the probes and analyzing the results. Probe selection is typically an offline, inexact, and computationally heavy process. Probing is an intrusive mechanism because it stresses the system with new requests. Also it is not guaranteed that the state of the system with respect to the failure being diagnosed has stayed constant till the time of the probe.

Multiprocessor system diagnosis: The traditional field of diagnosis has developed around multiprocessor systems, first addressed in a seminal paper by Preparata *et al.* [8] known as the PMC method. The PMC approach, along with several other deterministic models [21], assumed tests to be perfect and mandated that each entity be tested a fixed number of times. Probabilistic diagnosis, on the other hand, diagnoses 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[10]. Follow up work focused on multiple syndrome testing [9] 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. The authors in [22] propose a fully distributed algorithm that allows every fault-free node to achieve diagnosis in at most, $(log\ N)^2$ testing rounds. More recently, in [23] the authors extend traditional multiprocessor diagnosis to handle change of failure state during the diagnostic process. All of these approaches are fundamentally different from ours since there is no separation between the payload and the monitor system. This implies the payload system has to be observable *and* controllable (to generate the tests and analyze them).

Embedded system diagnosis: There has also been considerable work in the area of diagnosis of embedded systems, particularly in automotive electronic systems. In [24], the authors target the detection and shut down of faulty actuators in embedded distributed systems employed in automotives. The work does not consider the fallout of any imperfection in the analytical model of the actuator that gives desired behavior. The authors in [25] use assertions to correlate anomalies from the components to determine if a component is malfunctioning. The technique has some shared goals with the Monitor system – ability to trace correlated failures of nodes in a distributed system and handle non binary results from tests. The approach uses assertions that can examine internal state of the components. The papers in this domain do not consider imperfect observability of the sensor input or the actuator output, possibly because of tight coupling between the components. They are focused on scheduling monitor processes under processing resource constraints while we do not have such constraints.

Debugging in distributed applications: There has been a spurt of work in providing tools for debugging problems in distributed applications – performance problems [26]-[28], misconfigurations [29], etc. The general flavor of these approaches is that the tool collects trace information at different levels of granularity (line of code to process) and the collected traces are automatically analyzed, often offline, to determine the possible root causes of the problem [13]. For example, in [26], the debugging system performs analysis of message traces to determine the causes of long latencies. The goal of these efforts is to deduce dependencies in distributed applications and flag possible root causes to aid the programmer in a manual debug process, and not to produce automated diagnosis.

Automated diagnosis in COTS systems: Automated diagnosis for blackbox distributed COTS components is addressed in [30][31]. The system model has replicated COTS application components, whose outputs are voted on and the minority replicas are considered suspect. This work takes the restricted view that all application components are replicated and failures manifest as divergences from the majority. In [14], the authors present a combined model for automated detection, diagnosis, and recovery with the goal of automating the recovery process. However, the failures are all fail-silent and no error propagation happens in the system, the results of any test can be instantaneously observed, and the monitor accuracy is predictable.

In none of the existing work that we are aware of, there exists a rigorous treatment of the impact of the monitoring system's constraints and limited observability of the payload system on the accuracy of the diagnosis process. It is asserted in [26] that drop rates up to 5% do not affect diagnosis accuracy without supported reasoning.

7 Conclusion

In this paper we have presented the design and implementation of a distributed Monitor system for diagnosis of faults in distributed applications through observation of external messages among the protocol entities. We have presented a probabilistic model that accounts for deployment conditions and resource constraints at the Monitor. The parameters of the model can be automatically adjusted based on observed events by the Monitor. The Monitor system is demonstrated on a reliable multicast application and is shown to have a high accuracy and low latency for a wide range of deployment conditions.

Our current work is looking at collusion model or correlated failures among the PEs and automating the positioning and mapping of the Monitors to the PEs to verify. We are also conducting experiments with different applications.

References

[1] 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 '04), pp. 15-24, March 2004.

[2] G. Khanna, P. Varadharajan, and S. Bagchi, "Self Checking Network Protocols: A Monitor Based Approach," In Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems (SRDS '04), pp. 18-30, October 2004.

[3] META Group, Inc., "Quantifying Performance Loss: IT Performance Engineering and Measurement Strategies," November 22, 2000. Available at: http://www.metagroup.com/cgi-bin/inetcgi/jsp/displayArticle.do?oid=18750.

[4] 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 Proceedings of INFOCOM '02, pp.1209-1217, 2002.

[5] M. Castro, and B. Liskov,, "Practical Byzantine fault tolerance and proactive recovery," ACM Transactions on Computer Systems, vol. 20, no.4, 398–461, Nov. 2002.

[6] M. Correia, N. F. Neves, and P. Veríssimo, "How to tolerate half less one Byzantine nodes in practical distributed systems, " In Proceedings of 23rd International Symposium of Reliable and Distributed Systems, pp. 174–183, Oct. 2004.

[7] M. Correia N. F. Neves, and P. Veríssimo, "The design of a COTS real-time distributed security kernel," In Proceedings of the Fourth European Dependable Computing Conference, pp. 234–252, Oct.2002.

[8] F.P. Preparata, G. Metze, R.T. Chien. "On the Connection Assignment Problem of Diagnosable Systems"., " In IEEE Transactions on Electronic Computers, vol. 16, no. 6, pp. 848-854, Dec. 1967.

[9] D. Fussel and S. Rangarajan, "Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity," 19th Int. IEEE Symp. on Fault-Tolerant Computing, pp. 560-565, 1989.

[10] S. Lee and K. Shin, "Probabilistic diagnosis of multiprocessor systems," ACM Computing Surveys (CSUR), vol. 26, Issue 1, 1994.

[11] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," Proceedings of the 2002 IEEE Symposium on Security and Privacy, May 12-15, 2002.

[12] H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion Detection Alerts," Proceedings of the 4th Symposium on Recent Advances in Intrusion Detection (RAID 2001), Davis, CA, USA, Springer LNCS 2212, pages 85-103, October 2001.

[13] X. Y. Wang, D. S. Reeves and S. F. Wu, "Tracing Based Active Intrusion Response," In Journal of Information Warefare, vol. 1, no. 1, September 2001.

[14] K. R. Joshi, W. H. Sanders, M. A. Hiltunen, R. D. Schlichting, “Automatic Model-Driven Recovery in Distributed Systems,” At the 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05), pp. 25-38, 2005.
[15] A. T. Bouloutas, S. Calo, and A. Finkel, “Alarm correlation and fault identification in communication networks,” IEEE Transactions on Communications, vol. 42, pp. 523--533, 1994.
[16] B. Gruschke, “Integrated Event Management: Event Correlation Using Dependency Graphs,” at the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), pp. 130-141, 1998.
[17] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, “A coding approach to event correlation,” Intelligent Network Management, pp. 266-277, 1997.
[18] A. Frenkiel and H. Lee, “EPP: A framework for measuring the end-to-end performance of distributed applications,” in Proc. Performance Engineering Best Practices Conference, 1999.
[19] I. Rish, M. Brodie, and S. Ma, "Intelligent probing: A cost-efficient approach to fault diagnosis in computer networks," IBM Systems Journal, vol. 41, no. 3, pp. 372-385, 2002.
[20] I. Rish, M. Brodie, M. Sheng, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez, “Adaptive diagnosis in distributed systems,” IEEE Transactions on Neural Networks, vol. 16, no. 5, pp. 1088-1109, 2005.
[21] R. W. Buskens and R. P. Bianchini, Jr., “Distributed on-line diagnosis in the presence of arbitrary faults,” at the The Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS-23), pp. 470-479, 1993.
[22] E. P. Duarte, Jr. and T. Nanya, ”A hierarchical adaptive distributed system-level diagnosis algorithm,” IEEE Transactions on Computers, vol. 47, no. 1, pp. 34-45, 1998.
[23] S. Arun and D. M. Blough, “Distributed diagnosis in dynamic fault environments,” IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 5, pp. 453-467, 2004.
[24] N. Kandasamy, J. P. Hayes, and B. T. Murray, “Time-constrained failure diagnosis in distributed embedded systems,” at the International Conference on Dependable Systems and Networks (DSN), pp. 449-458, 2002.
[25] P. Peti, R. Obermaisser, and H. Kopetz, “Out-of-norm assertions [diagnostic mechanism],” at the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), pp. 280-291, 2005.
[26] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, “Performance debugging for distributed systems of black boxes,” at the Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA, pp. 74-89, 2003.
[27] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, “Pinpoint: problem determination in large, dynamic Internet services,” at the International Conference on Dependable Systems and Networks (DSN), pp. 595-604, 2002.
[28] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, “Magpie: online modelling and performance-aware systems ” at the 9th Workshop on Hot Topics in Operating Systems (HotOS IX), pp. 85-90, 2003.
[29] H. J. Wang, J. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, “PeerPressure for automatic troubleshooting,” at the Proceedings of the joint international conference on Measurement and modeling of computer systems, New York, NY, USA, pp. 398-399, 2004.
[30] A. Bondavalli, S. Chiaradonna, D. Cotroneo, and L. Romano, “Effective fault treatment for improving the dependability of COTS and legacy-based applications,” IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 4, pp. 223-237, 2004.
[31] L. Romano, A. Bondavalli, S. Chiaradonna, and D. Cotroneo, “Implementation of threshold-based diagnostic mechanisms for COTS-based applications,” at the 21st IEEE Symposium on Reliable Distributed Systems, pp. 296-303, 2002.
[32] http://www.experimentalstuff.com/Technologies/JRMS/
[33] InternetWeek 4/3/2000 and “Fibre Channel: A Comprehensive Introduction,” R. Kembel 2000, p.8. Based on a survey done by Contingency Planning Research.
[34] A. Brown and D. A. Patterson, “Embracing Failure: A Case for Recovery-Oriented Computing (ROC),” 2001 High Performance Transaction Processing Symposium, Asilomar, CA, October 2001.
[35] G. Khanna, P. Varadharajan, M. Cheng, and S. Bagchi, “Automated Monitor Based Diagnosis in Distributed Systems,” Purdue ECE Technical Report 05-13, August 2005. Also in submission to IEEE Trans. on Dependable and Secure Computing.
[36] http://min.ecn.purdue.edu/~gkhanna/Rules.html.

Appendix

A. Abbreviations

CG	Causal Graph	PPEP	Path Probability of Error Propagation
AG	Aggregate Graph	PE	Protocol Entity
IM	Intermediate Monitor	LM	Local Monitor
GM	Global Monitor	TL	Temporary Links
DT	Diagnosis Tree	SRB	Strict Rule Base (Diagnosis)
LC	Logical Clock	NRB	Normal Rule Base (Detection)
SS	Suspicion Set	TTCB	Trusted Timely Computing Base
EMC	Error Masking Capability	TRAM-D, TRAM-L	Tram Distributed and TRAM Local.