'STRESS' Testing Applied to a Multicast Routing Protocol

Ahmed Helmy, Deborah Estrin
Computer Science Department/ISI
University of Southern California
Los Angeles, CA 90089
email:{ahelmy, estrin}@usc.edu *

July 22, 1997

Abstract

Multiparty protocols support an important class of applications ranging from multi-media teleconferencing to network games. Designing wide-area multiparty protocols is becoming more complex with the growth of the Internet and the introduction of new service models. Unexpected combinations of events can drive protocols into undesirable states and may lead to errors. Anticipating all such cases is often impossible and at best may require extensive simulation and testing. In large systems, the cost of testing all possible scenarios exhaustively is prohibitive, and many unexpected cases are not observed until deployment.

Prototyping in testbeds or individual simulations typically focuses on performance under a limited set of randomized protocol transitions. Formal and analytical models representing such protocols tend to be complex, sometimes rendering the model intractable.

In this work, we propose a method for analyzing the robustness of multiparty (multicast-based) protocols in a systematic fashion. We call our method *Systematic Testing of Robustness by Examination of Selected Scenarios (STRESS)*. STRESS aims to cut the time and effort needed to explore the pathological cases of a protocol during its design. This paper has two goals: (1) to describe the method, and (2) to serve as a case study of robustness analysis of multicast routing protocols. We do not prove correctness but aim to offer design method tools similar to those used in CAD and VLSI design.

1 Introduction

In this paper, we describe a method for **S***ystematic* **T***esting of* **R***obustness by* **E***xamination of* **S***elected* **S***cenarios (STRESS)*. It is based on a simulation framework supported by a set of tools, and is designed for studying protocol behavior in the context of pathological cases and scenarios. Some of the general concepts for STRESS draw from protocol verification techniques and reachability analysis [1, 2]. We apply these techniques to support the design, analysis, and testing of *multiparty protocols*.

In particular, we introduce techniques for state and topology reduction and investigate various packet loss scenarios to capture robustness characteristics. The definition of *error conditions* enables us to capture the *faulty (error-prone)* cases automatically.

Multiparty protocols may involve multiple receivers and one or more senders. These protocols include multicast routing protocols (e.g. DVMRP [3], MOSPF [4], PIM-DM [5], CBT [6], and PIM-SM [7]), multicast transport protocols (e.g. SRM [8], RTP, and RTCP [9]) and multiparty applications (e.g. WB [10], vat [11], vic [12], nte [13], and sdr [14]). This paper focuses on multicast routing protocols, which deliver packets efficiently to group members by establishing distribution trees, as shown in figure 1. As a case study, we apply our method to the multicast routing protocol, Protocol Independent Multicast-Sparse Mode (PIM-SM).

*This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract

Figure 1: Establishing multicast delivery tree

We uncovered several pathological errors in PIM-SM through the use of STRESS tools, and evaluated solutions to eliminate these errors. The suggested solutions have since been added to the PIM-SM specification [15].

The rest of the paper is organized as follows. Section 2 provides an overview of the STRESS method. The case study for PIM-SM is presented in section 3. Results are given in section 4. Sections 5 and 6 address related work, summary and future work, respectively.

2 The Approach–Method Overview

The *robustness* of a protocol is its ability to respond correctly in the face of network failures and packet loss. The goal of the *STRESS* method is to provide a framework for systematic testing of protocol robustness through the examination of selected scenarios.

For a given protocol, we first capture a set of error-prone scenarios. This is achieved by: (a) investigating a *representative* subset of the protocol state space, and (b) defining error conditions. We use these scenarios to evaluate design trade-offs, analyze behavior, and as a test suite to examine various implementations of the protocol.

Our basic approach consists of three stages: *scenario generation* (pre-processing), *tracing* (simulation), and *output analysis* (post-processing), see figure 2.

2.1 Scenario Generation

Scenarios are composed of routed topologies and sequences of events (input stimuli and state transitions), and describe the simulation context that may cause protocol transitions. Scenario parameters include the *routed topology*, *host scenarios* and *loss scenarios.*

2.1.1 Routed topology

The routed topology is the network infrastructure upon which the protocol operates; e.g. network nodes and links and the unicast routing that determines how packets are forwarded.

No. DABT63-96-C-0054. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

Figure 2: *STRESS* method block diagram

We try to identify simple topologies that capture a large percentage of the protocol's state space, and to which other, more complex, topologies may be reduced[1]. For much of our study, we choose a LAN with four connected routers as the basic topology. We show how other topologies are reducible to the *four-router* LAN topology, and discuss the limitations of such a topology in section 3.2. We further extend the topology to capture particular characteristics of the protocol under study, PIM-SM.

As a component of the routed topology, unicast route inconsistencies may be a common source of error. Unicast routing may exist in one of the following three states: (a) consistent routing, (b) transient inconsistent routing, and (c) long lived inconsistency. Case (a) requires no changes. The study of case (b) is convergence analysis, which has been addressed elsewhere[2]. We are particularly interested in case (c)[3]. We add an inconsistent unicast routing component to force the multicast routing protocol into states encountered in such pathology, and analyze those states.

2.1.2 Host scenarios

Host scenarios are combinations of possible host actions. In our case study, these are defined by the multicast service model. Host actions include joining (or leaving) groups, or sending packets to groups. For large numbers of hosts and groups it is prohibitively costly to explore all possible combinations exhaustively.

The simplest multicast host scenario has a single source 'S' and two receivers 'R1' and 'R2' for the same group. We shall address this simple scenario in this section, and show (in section 3) how it can be utilized and extended for our case study.

We estimate all the possible combinations of our host model, and try to reduce the number to those scenarios that may affect the protocol state transitions. We call such scenarios *representative scenarios*. To obtain the representative scenarios we apply the *scenario filter* shown in figure 3.

For one source and two receivers, the five possible host events are: source 'S' sending to a group (or 'S' for short), receiver joining a group (or 'J1' and 'J2' for receivers 'R1' and 'R2', respectively), and receiver leaving a group (or 'L1' and 'L2' for receivers 'R1' and 'R2', respectively).

For all possible permutations, there exists $5! = 120$ scenarios, considering that each host event occurs once. Applying protocol constraints, such as *a receiver cannot leave before joining the group*, reduces the number of possible combinations to $5!/(2! \times 2!) = 30$ scenarios. Further assuming for practicality, without

[1]Two topologies are said to be reducible (or equivalent) if they drive the protocol (according to the host scenarios applied) into the same states, experiencing the same set of state transitions.

[2]For convergence analysis of PIM-SM mechanisms, refer to [16].

[3]This may be caused by a multicast region spanning more than one unicast routing AS.

Figure 3: The Scenario Filter to obtain representative scenarios

loss of generality, that *the source sends packets throughout the simulation*, reduces the number of possible scenarios to $30/5 = 6$ scenarios. These six scenarios are:

1. `J1:J2:L1:L2`
2. `J1:J2:L2:L1`
3. `J1:L1:J2:L2`
4. `J2:J1:L1:L2`
5. `J2:J1:L2:L1`
6. `J2:L2:J1:L1`

The number of representative scenarios can be even reduced further if the host distribution is symmetric with respect to the topology, since the following scenarios will be equivalent: (i) 1 equivalent to 5, (ii) 2 equivalent to 4, and (iii) 3 equivalent to 6; i.e. we need only investigate 3 different host scenarios for the given topology.

These scenarios may be generated automatically by the method. However, generalizing the process of obtaining representative scenarios for various multiparty protocols is currently under study.

2.1.3 Loss and Failures

The loss and failure scenarios considered include the loss and corruption of packets (during transport, routing or forwarding within the network), or loss of state in router nodes due to protocol daemon failures, machine crashes, or insufficient resources (e.g. memory).

Loss of packets Packet loss may occur in various parts of the network, due to congestion, or link, node or interface failures. We classify these events as simply 'packet loss' regardless of cause, and create exhaustive loss scenarios to capture all the possible protocol transitions and pathologies due to packet loss.

For most multicast protocols, when routers are connected via a multi-access network (or LAN)[4], hop-by-hop messages are multicast on the LAN, and may experience selective loss; i.e. may be received by some nodes but not others. The likelihood of selective loss is increased by the fact that LANs often contain hubs, bridges, switches, and other network devices. Selective loss may affect protocol robustness. Similarly, multiparty protocols and applications must deal with situations of selective loss. This differentiates these applications most clearly from their unicast counterparts, and raises interesting robustness questions.

Our case study illustrates why selective loss should be considered when evaluating protocol robustness. This lesson is likely to extend to the design of higher layer protocols that operate on top of multicast and can have similar selective loss.

[4] We use the term LAN to designate a connected network with respect to IP-multicast. This includes shared media (such as Ethernet, or FDDI), hubs, switches, etc.

The input to the 'loss & failures' substage (shown in figure 2) is obtained from initial traces of simulations without protocol message loss. These traces guide further simulations to cover all possible protocol message loss scenarios.

Loss of state State loss in nodes may occur due to crashes, loss of multicast, unicast or all forwarding entries. We investigate how such loss affects the protocol's overall correctness, especially from the end-systems perspective.

2.2 Simulation and Tracing

During this stage the protocol mechanisms are simulated and traces are collected:

2.2.1 Simulation

One desirable approach for simulating complex protocols, is to include detailed mechanisms of parts of the protocol while abstracting out others, we call this approach *subsetting*.

Subsetting refers to selecting subsets of the protocol functions, while abstracting or removing others. This allows us to focus on specific parts of the protocol state space. Subsetting can be based on:

- *Protocol functions*: Subsetting protocol functions (or mechanisms) refers to the abstraction of these functions. This may be achieved by replacing a complex mechanism by a simpler one, exhibiting similar external behavior under relaxed assumptions. For example, one may use static configuration instead of simulating a detailed bootstrap algorithm. This way, one may study other protocol mechanisms assuming correctness of the bootstrap mechanism.
- *Protocol states*: A study may focus on specific protocol states. This allows, for example, the study of multicast group state without dealing with source-specific state.
- *Messages types*: This allows the examination of specific protocol message types in the absence of others.

Note that subsetting does not permit all combinations of the above items. To maintain protocol correctness, an abstracted part has to be replaced by its equivalent that exhibits similar external behavior. For example, one may not simply remove the bootstrap mechanism and run the simulations.

2.2.2 Tracing

Tracing is the process of logging information about events or packets during the simulation run. Logged information is analyzed during the post-processing (i.e. the output analysis) stage. In addition, some traces are used as feedback to the scenario generator to guide further simulations. We consider several kinds of tracing:

End-point tracing Tracing end-points includes logging information pertaining to hosts sending or receiving packets, and joining or leaving multicast groups. A detailed description of the traces used in the case study is given in section 3.

To identify errors and pathologies in the protocol itself, we focus on the effect of the multicast routing protocol transitions on the end-point packet delivery (as explained in section 2.3).

Protocol state transition tracing A protocol can be represented by a finite state machine (automaton), consisting of states, transitions and stimuli (inputs, outputs, and timer actions). Based on knowledge of initial protocol states, we obtain the sequence of protocol transitions by tracing all stimuli.

We use protocol traces to diagnose and verify protocol behavior, and to analyze errors.

Link tracing We keep track of packets traversing links (or LANs) between nodes, as well as the events of packet loss on links (or LANs). This information is used in several ways: in output visualization, output analysis, or as feedback for scenario generation. Links carrying message types of interest are targeted for intentional packet loss in further simulations. This reduces the number of loss scenarios examined to those directly affecting the protocol behavior under investigation.

Code annotation When placed in key points (such as beginning of protocol procedures or code modifying the state of the protocol), code annotations capture internal execution of the protocol machinery. We use code annotation to estimate what part of the code, and subsequently the protocol, has been executed and stressed (code coverage).

2.3 Output Analysis

One major concern of STRESS is to identify pathological cases, and indicate when and if an error occurred and why. This is achieved in the output analysis stage, which consists of:

Identifying end-point errors *Error conditions* may be specified with respect to end-point traces. Examples of end-point error conditions are black holes[5], and packet duplication, where more than one copy of the same packet is received by a group member.

If the factors during one simulation run are relatively static (i.e. static unicast routing, static topology and controlled loss), the error may be attributed to an error in the multicast routing protocol.

Once the specified error is identified by the output analyzer, the trace log is rolled back in time to investigate the protocol traces, as explained next.

Relating errors to protocol After detecting an end-point error, the output analyzer isolates the possible causes of such errors in the form of protocol traces. The output analyzer in this case is similar to a logic analyzer, allowing the designer to navigate backward in time and investigate the causes of the error.

As will be shown (in section 3), the process of identifying a protocol error may suggest fixes to the problem.

Code profiling The profiler captures information about the annotated code, such as which procedures were (or were not) invoked, and the order and frequency of invoking protocol procedures. This information indicates the portion of the protocol stressed by the examined scenarios.

3 Case Study of PIM-SM

To evaluate the utility of STRESS, we applied it to a complex multicast routing protocol; PIM-SM. Before going into details of the case study, we first give an overview of multicast routing and PIM-SM.

3.1 Multicast Routing – Overview

Multicast distribution trees may be established by either broadcast-and-prune or explicit join protocols. In the former, such as DVMRP or PIM-DM, a multicast packet is broadcast to all leaf subnetworks. Subnetworks with no local members for the group send *prune* messages towards the source(s) of the packets to stop further broadcasts. Link state protocols, such as MOSPF, broadcast membership information to all nodes. In contrast, in explicit join protocols, such as CBT or PIM-SM, routers send hop-by-hop *join* messages for the groups and sources for which they have local members. When received, these messages build routing state in routers, and cause further messages to be sent upstream until the distribution tree is established. Upon arriving at a router, a multicast packet is forwarded according to the routing state.

[5]A black hole is an observable amount of consecutive packet loss between periods of packet delivery.

Figure 4: How senders rendezvous with receivers (*simplified*)

In this paper we study PIM-SM's mechanisms for building shared multicast trees. For simplicity, we do not address source-specific trees in this description.

As shown in figure 4, when a receiver's local router (A) discovers it has local receivers, it starts sending periodic *join* messages toward a group-specific Rendezvous-Point (RP). The *join* messages are multicast hop-by-hop. Each router along the path toward the RP builds a wildcard (any-source) *route entry* for the group and sends the *join* messages on toward the RP. A *route entry* is the state held in a router to maintain the distribution tree. Typically it includes the source address, group address, the interface from which packets are accepted (*incoming interface*), and the list of interfaces to which packets are sent (*outgoing list*). This state forms a shared, RP-rooted, distribution tree that reaches all group members.

When a source first sends to a group, its local router (D) unicasts *register* messages to the RP with the source's data packets encapsulated within. Data packets reaching the RP are forwarded natively down the shared tree toward group members.

Similarly, when a member leaves the group, a *prune* message is sent by the local router, to stop the multicast traffic from flowing down the branch leading to the pruned member.

Being robust to (at least) a single message loss, was a design goal for PIM-SM.

We point out two PIM mechanisms relevant to this study; *Assert* and *prune-override*. The PIM *Assert* mechanism is the process by which at most one forwarder for a LAN is selected to avoid duplicates in case of multiple potential forwarders due to parallel paths to the source or RP. The *prune-override* enables a downstream router (i.e. with downstream members) to retain its established branch of the tree, in case another router on the same LAN tries to prune that branch[6].

The rest of this section is outlined as follows. Section 3.2 establishes the equivalence relationship for the topology used for the case study. Section 3.3 describes the simulation test suites. An example of applying the method is explained in section 3.4, and the scenario and protocol coverage achieved for the case study is given in section 3.5.

3.2 Topology Equivalence

Two topologies are equivalent if they drive the protocol transitions into the same states, under the same set of event sequences. A topology is reducible to another topology –with fewer connections and routers– if the two topologies are equivalent.

[6]To achieve this, a downstream router receiving a *prune* on its incoming interface triggers a *join* upstream.

Figure 5: The equivalent topologies

We show in this section that for single message loss scenarios, the *four-router* LAN topology adopted in this study experiences the same protocol errors that an *N-router* LAN topology experiences –where $N > 4$–, and hence they are equivalent. We show this relationship for the messages under study for PIM-SM; namely, *joins*, *prunes* and *asserts*. For brevity, we only prove equivalence in the case of *prune* messages, and hint to the proof approach in the other cases. We also identify assumptions and limitations under which this equivalence relationship holds.

3.2.1 Prunes

First, we consider *N-router* LAN topologies, where $N = 1, 2$, and 3, respectively. It is trivial to prove that these topologies are not equivalent for hop-by-hop messages.

Assumption *N-router LAN topology, where $N > 3$, is reducible to the three-router LAN topology for prunes, w.r.t. single message loss scenarios.*

To justify our assumption we first prove that a *four-router* LAN topology is reducible to a *three-router* LAN topology.

Correctness condition: *If a router on the LAN has the LAN as its incoming interface, there must be one other router with the LAN in its outgoing list.* Once this condition is satisfied, violating it is considered a protocol error[7].

Next, we examine the *three-router* LAN topology. In figure 5, topology 'I', assume that A and B are downstream routers, and C is an upstream router.

- In figure 5, topology 'I', router C has the LAN in its outgoing list, router A has the LAN as its incoming interface, and router B is leaving the group and so sends a *prune* towards C. The *prune* is multicast on the LAN.

  The only case where the correctness condition may be violated is when C receives the *prune* while A does not. In the other cases, either the *prune* is not received by C, or is received by A which triggers a *prune-override* to re-establish the LAN in C's outgoing list. This is illustrated by the following selective loss pattern table for the *prune* message sent by B:

[7] This is to differentiate between join latency (which is not considered a protocol error) and a black hole which is a protocol error.

A	C	
0	0	
0	1	
1	0	← error
1	1	

where a '0' indicates no-loss and '1' indicates loss. The error occurs where the upstream router (C) received the *prune*, but the router with downstream members (A) did not receive it.

- In figure 5, topology 'II', we add another downstream router D. The selective loss pattern table follows:

A	D	C	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	← error
1	1	1	

The only error occurs when the upstream router (C) receives the *prune*, but neither of the downstream routers receives it. If the *prune* is received by any of the downstream routers, a *prune-override* would re-establish the LAN in C's outgoing list.

From the symmetry of the loss patterns and topology we see that all errors are triggered by the same transitions experienced by router A in topology 'I'. Hence, the extended topology 'II' does not introduce any new errors, and exhibits the same external behavior as does topology 'I'. We conclude that topology 'I' and topology 'II' are equivalent for *prunes*.

We now show that the *N+1-router* LAN topology is reducible to the N case; where $N \geq 3$.

With the addition of an upstream router (figure 5, topology 'III'), no added error cases are encountered. The addition of a downstream router, however, may introduce new error scenarios. Similar to the *four-router* LAN case, we establish the following assertion: *'the only error case occurs when all downstream routers lose the prune and the upstream router receives it.'* If the *prune* was received by any of the downstream routers, the correctness condition would be retained using *prune-overrides*.

The assertion holds in both topologies. Hence, we conclude that the *N-router* topology experiences the same errors as the *N+1-router* topology.

From the above we see that by simulating the *three-router* LAN topology we capture all the errors, with respect to selective loss (for the *prune* mechanism), that may be experienced by any *N-router* LAN topology; where $N > 3$.

3.2.2 Joins

For loss-free scenarios, the equivalence proof is straightforward. For lossy scenarios, the loss of a *join* message sent to an upstream router may lead to join latency, but does not cause black-holes[8]. Joins leading to packet duplication lead to *asserts* that are discussed next.

[8]Note that if join suppression as described by the PIM-SM specification [15] is implemented, the equivalence relation can be established similar to the *prunes* case above.

Figure 6: The topology used for the case study

3.2.3 Asserts

In most cases, proofs similar to those presented can be applied to *Asserts*. However, since asserts may be triggered due to parallel paths, the base case is established for the *four-router* LAN topology. Figure 5, topology 'III', represents the *four-router* LAN topology, where A and B are downstream routers, and C and D are upstream routers[9].

For our case study, we use a *four-router* LAN topology with an added Rendezvous-Point (RP) to capture shared tree characteristics. The overall physical topology consists of five routers, four of which are connected via a LAN, as shown in figure 6.

3.3 Test suites

In this section we elaborate on the routed topology, host scenarios and loss pattern generation used for our case study. We also describe the simplifications and subsettings applied.

Physical and routed topologies The overall topology used is that shown in figure 6. For the unicast routing protocol we use a centralized version of Dijkstra's Shortest Path First (SPF) algorithm [17].

PIM-SM uses the underlying unicast routing tables for building multicast trees. Therefore, unicast routing inconsistencies affect the operation of PIM-SM. To investigate such interaction we add a component to force inconsistent multicast routes between PIM routers, as shown in figure 6, topology 1. Inconsistent unicast routing cases arise, for example, when a multicast region spans multiple unicast domains (or ASs)[10].

Host scenarios Since protocol states for different groups do not interact, we consider only one group. Also, since protocol states for different sources do not interact, it suffices to consider only one source 'S' per simulation run[11]. The source is modeled as a constant-bit rate (CBR) stream with fixed packet size. The

[9] A limitation to the *four-router* LAN topology is given for the esoteric case of three upstream routers and three downstream routers with inconsistent unicast routing tables. This case creates one extra transition that can only be captured by at least a *six-router* LAN topology. We do not consider this a practically significant scenario, and we consider its analysis as a special case, not captured by the *four-router* LAN topology.

However, aside from this exception, the *N-router* LAN topology (where $N > 4$) is equivalent to a *four-router* LAN topology w.r.t. *asserts*.

[10] A similar interaction with unicast routing may be created easily when switching to the shortest paths in PIM-SM.

[11] We do not consider aggregated source or group entries in this study.

source model does not affect the correctness of the method. However, to assure full controllability over the selective loss model, we set the data rate to ensure that no loss occurs due to queue overflow[12].

While we consider only a single source, we consider two receivers ('R1' and 'R2') for the same group to account for shared tree state interactions. We use the host scenarios described in section 2.1.2.

Loss patterns We investigate all possible selective loss scenarios for multicast hop-by-hop PIM-SM messages in this representative topology.

Loss models are applied exhaustively to those links that carry the protocol messages under investigation. The tracing stage identifies these links during the first simulation run, without packet loss, and feeds back the link information to the loss generation module, as shown in figure 2. As we will show in section 3.5, the number of representative scenarios is quite small, and hence the number of overall lossy scenarios explored is manageable.

We do not address state loss or node crashes in this document. However, crash scenarios may be implemented in a way similar to loss scenarios.

Tracing Trace information includes the event type (send or receive), the node experiencing the event, the type of message sent or received, and the time at which the event occurred. Every data packet is assigned a unique sequence number.

For example, a trace may take the following format: '**R2 Node A Rcvd 7 time 190**' meaning that receiver **R2** in node **A** received a data packet with sequence number **7** at time **190ms** from the beginning of the simulation run.

Subsetting For brevity, we do not consider source-specific trees and switching to the shortest paths in this paper. This is an example of *state subsetting*, since we consider shared group states while disregarding source-specific states.

The messages considered in the study are *join*, *prune*, *assert* and *register* messages. To study *joins*, *prunes* and *asserts* without the effect of *registers*, we consider a topology where the source and the RP are co-located (see S1 in figure 6, topology 1). This is an example of *message subsetting*.

When studying *registers*, *joins* and *prunes* we consider topology 2 in figure 6 where: (a) S2 is the source, hence node A sends registers to the RP, and (b) the routed topology has consistent unicast routing, to eliminate the effect of the *assert* mechanism. This represents *function (or mechanism) subsetting*.

Only triggered actions are investigated for simplicity. Timer action analysis is not considered in this study, and is part of our work-in-progress.

3.4 Applying the Method

This section provides an illustrative example, showing how STRESS may be used to identify and analyze errors encountered during the simulation of the representative scenarios.

We have implemented an initial version of the STRESS method in the Network Simulator 'NS' [18]. NS is an event-driven packet-level simulator controlled and configured via Tcl [19] and Object-Tcl (or OTcl) [20][13]. To support our method, we have added Modules to provide LAN support, controlled selective loss, protocol tracing, profiling capabilities, and a detailed implementation of PIM-SM[14]. This implementation serves as the simulation environment for our case study. In addition, the building blocks were designed to be re-used within the same framework to apply this method to other multiparty protocols.

[12] For this we use packet size of 180 bytes, and a send interval of 25 ms (i.e. source rate of 57.6 kb/s), this ensures no queue drops on the 1.5 Mb/s links used with 10 packet queue limit.

[13] For information about the simulator see http://catarina.usc.edu/vint.

[14] Our detailed PIM-SM simulation mimics the unix 'pimd' [21] implementation model, and hence is able to capture many implementation aspects. We plan to develop an interface between the simulator and an operational network running the pimd code. However, the analyses presented in this study are based strictly on the protocol specification, independent of the implementation.

Figure 7: Simple packet trace graph showing packet loss and duplication

Obtaining faulty scenarios To obtain the *faulty scenarios* (i.e. those that contain errors), we execute the method stages in order (i.e. scenario generation, simulation and tracing, and output analysis, respectively), and then reverse the order from the output to the traces to identify the faulty scenarios. These phases are automated by the tools provided, and are transparent to the user, once the scenario set-up is complete.

The process of attributing end-point errors to protocol actions may be automated only if the error conditions are given in terms of such protocol actions. In practice, however, these protocol error conditions are often not known *a priori* by the designer, and are usually defined in terms of end-point errors (such as packet loss or duplication). The supporting tools identify end-point errors, and provide a history of protocol traces. The designer then examines the traces and identifies the protocol errors[15]. This process may suggest fixes to the problem, as we will show in the results section.

Example In our simple example, an *error condition* is any packet loss or duplication, experienced by the end-points. A faulty scenario (without packet loss) that leads to two error conditions, is identified and explained. Then the protocol actions leading to the errors are analyzed.

The representative scenario explained here is 'J1:J2:L1:L2' using topology 1. This scenario was identified automatically as a faulty scenario. Traces in figure 7 give the history of the errors found. The first error (i.e. the packet duplication) has the host event 'J2' as the closest join or leave host event in its history at time 200ms. The error is a join transient caused by parallel paths to the RP. The error is resolved using the *Assert* messages exchanged during the duplication at time 246ms. The second error (i.e. packet loss) is a leave transient; it has a host event 'L1' in its recent history. The loss is due to the *prune* sent by node A at 300ms, and is resolved by a *prune-override* sent by node B at 310ms.

Although the protocol actions leading to the end-point errors (specified as any packet loss or duplication in this specific example) are considered transient errors, they are not considered protocol design errors. We do, however, address protocol design errors in section 4.

3.5 Scenario and protocol coverage

While the fact that we were able to discover design errors provides some evidence of the method's utility, we would like to quantify the coverage of protocol states and possible scenarios.

The overall protocol coverage has two dimensions. The first is the protocol state coverage, and we attempt to cover this dimension using the *representative* scenarios' reachable states. Investigation of the *loss* scenarios

[15]In our limited experience, we have identified protocol errors in the recent history of the end-point errors.

does not affect protocol coverage significantly.

The second dimension is the space of possible interaction scenarios between these state machines (in different routers) within the topology. This dimension is explored by investigating the *selective loss* scenarios.

Scenarios covered The initial number of simulated scenarios *without* protocol message loss was:

$$\sum_{topologies} (\text{No. rep. scenarios})$$

Where No. rep. scenarios is the number of *representative* scenarios, equal to 6 in our case (discussed in section 2.1.2), and the topologies are the two discussed in section 3.3. Hence we simulated 12 scenarios without protocol message loss.

After feeding back the link traces for the messages under study, the loss patterns were assigned to the corresponding links. The scenario generator then set-up the simulations for the new scenarios with loss.

The total number of scenarios *with* protocol message loss simulated is given by the following formula:

$$\sum_{\forall Topos} \left(\sum_{\forall Reps} \left(\sum_{\forall Msgs} \left(\sum_{\forall Links} LinkMsgs \cdot 2^{(LinkRtrs-1)} \right) \right) \right)$$

where,

Term	Meaning
Topos	Topologies
Reps	Representative Scenarios
Msgs	Messages under study
LinkMsgs	Number of messages traversing the link
LinkRtrs	Number of routers connected to the link

For example, for the first topology, the messages under study were *joins*, *prunes* and *asserts*. The representative scenarios triggered 16 *joins*, 12 *prunes*, and 12 *asserts* on the LAN, and 16 *joins* and 16 *prunes* on point-to-point links. For the second topology, the messages under study were *joins* and *prunes*. The representative scenarios triggered 16 *joins*, and 18 *prunes* on the LAN, and 6 *joins* and 18 *prunes* on point-to-point links. Hence, the total number of scenarios with loss became 352 and 296 scenarios, respectively.

Protocol code coverage A large portion of the multicast support code in *ns* was annotated automatically to provide code tracing. The representative scenarios without loss invoked 84 procedures out of 91 overall annotated procedures. The procedures that were not invoked dealt mainly with source-specific state (which was abstracted in our test suites), or with the modularity of the object-oriented nature of the code.

4 Results

This section describes the protocol design errors uncovered for PIM-SM under STRESS.

Unlike our simple example above, we are only interested in design (i.e. not transient) errors. For this, we modified the *error conditions* to avoid join and leave transients. The new *error conditions* do not consider single duplication or loss.

Following is a summary of the major faulty scenarios encountered, and how they relate to STRESS. For a more detailed discussion of the protocol errors and fixes see section 4.2.

4.1 Summary of Results

We describe a partial list of *faulty scenarios* captured by STRESS. We obtained this list after simulating only a few of the representative scenarios. The traces produced provided guidance to discover the protocol errors. Design errors discovered include *Assert*, *Join/Prune* and *Register* mechanisms.

Asserts For the first topology (figure 6, topology 1), a black hole was observed for one receiver.

The faulty scenario in this case involved another receiver joining in the recent history of the black hole. By analyzing the protocol trace history after rolling back, we noticed that an *Assert* process took place right before the loss.

In addition, the faulty scenario included the loss of a *join* message, which prevented the establishment of the branch of the shared tree from the Assert winner to the RP. Hence, the protocol design error is allowing a router on a branch of the tree that is not completely established, to participate in *Asserts*.

Joins and Prunes Over the same topology (i.e. figure 6, topology 1), several other faulty scenarios lead to black holes. The host scenarios involved one receiver leaving just before black holes were experienced by the other receiver. In these cases *join* and *prune* messages occurred the recent history of the end-point error.

Furthermore, all such scenarios included either: (i) loss of a *join* message, preventing a pruned branch from being re-established; or (ii) selective loss of a *prune* message, preventing a *join* (i.e. *prune-override*) from being triggered. The protocol design error in this case was not allowing a second chance for routers with downstream members to override *prunes*.

Registers In the second topology (figure 6, topology 2), faulty scenarios were captured that cause packet duplicates at the end-points.

In this case, the observed faulty scenarios did not follow a regular pattern, and were developed iteratively (i.e. when one faulty scenario led to a suggested fix in the protocol, the fix was implemented and the method re-run to observe further faulty scenarios).

The first scenario involved a single host receiving duplicates merely by joining the group. The packets were being delivered at least twice, once directly from the source –by virtue of being on the same LAN–, and the second delivery from the shared tree after the *register* reached the RP and was sent down the shared tree. When the number of packet duplicates exceeded two, this suggested a loop. The loop occurred when a packet received over the shared tree on the LAN, was (a) picked up by the local router, (b) re-registered to the RP, and (c) forwarded down the shared tree again. The protocol error was allowing the packets to flow down from the shared tree to the originating LAN, and be re-registered. The fix was to prune such sources from the shared tree.

The second scenario involved another receiver joining before the duplicates were observed. The pruned branch of the shared tree was re-established by the joining receiver, allowing the packets to flow down the shared tree to the originating LAN, and subsequently, causing the loop.

The third scenario involved a *prune* message loss, again allowing the packets to flow down the shared tree to the originating LAN, and led to looping.

Rules were added to prevent packets from being forwarded back on their original LANs in the above scenarios.

4.2 Detailed Results

The rest of this section describes the above faulty scenarios in more detail, and illustrates how the solutions were developed with the aid of STRESS.

Figure 8: The *Assert* scenario under study

4.2.1 Assert analysis

Following is a discussion of the pathological cases found in the *Assert process*. An exhaustive list of the results is not included in this document for brevity. A few errors in the PIM-SM specification were unveiled during this process, we focus on errors that created the possibility of packet loss (i.e. black holes).

The scenario In this scenario, the topology in figure 8 was set-up such that *A's* next-hop towards the *RP* is *C*, and *B's* next-hop towards the *RP* is *D*.

Consider the sequence of events shown in figure 8, which used the representative scenario 'J1:J2:L1:L2' with the loss of a *join* message on the link between *C* and *RP*.

During the last two events of the scenario (steps 6 and 7), *D* lost the *Assert* process to *C* (with higher metric or address). Subsequently, *D* removes the LAN from its entry's interface list, and R1 stops receiving packets from S1. This problem persists until/unless the branch of the tree from *C* to *RP* is established.

Discussion and fix The current rules of the PIM specification aim to guarantee 'at-most' one forwarder on a multi-access network. However, to ensure proper delivery of packets without packet loss, the right semantics should be 'exactly' one forwarder.

The problem arises, more specifically, because the PIM specification does not distinguish between an 'active' entry (i.e. an entry created due to arrival of data packets, e.g. a multicast forwarding cache), and an entry on a branch of a tree that is not yet established (or an *'inactive'* entry). An 'inactive' entry may win an *Assert* process, resulting in black holes.

To solve this problem, we modified the specification to ensure 'exactly' one forwarder semantics using the following rule:

Figure 9: A simplified state transition diagram for the *join* and *assert* processing

- A router receiving a data packet (or *Assert*) on an outgoing interface of a matching entry does not participate in the *Assert* process unless the entry is 'active'.

Figure 9 illustrates the 'ActiveState' added to the transition diagram to realize the solution.

4.2.2 Join/Prune analysis

In this analysis we address the effect of selective loss of *Join/Prune* messages. Although this problem has been addressed in recent releases of the PIM-SM specification, we provide a more efficient solution.

We use the topology given in figure 10, **I**. The representative scenario used is 'J1:J2:L1:L2' with the second *join* from node A lost on the LAN.

We assume that S1 sends packets to group G throughout the simulation. Consider the sequence of events given in figure 10, **I**. After the last event (step 5), R2 stops receiving S1's packets. This problem persists until A sends the next periodic *join* to C and re-establishes the pruned branch of the tree. A similar problem is encountered in figure 10, **II**, when the *prune* sent from B is selectively lost on the LAN by A and received by C.

Discussion and fix The solution suggested by the PIM specification introduces a deletion timer. This, however, increases the leave latency, and incurs unnecessary data overhead.

A more efficient solution would be to have the upstream router (C) announce a *'prune-alert'*, before removing the LAN from its outgoing list, by resending the *prune* message previously received from B.

4.2.3 Register analysis

Following is a description of the scenarios that exhibit packet duplication due to *register* messages, and the suggested fixes to eliminate such duplication. The fixes were applied iteratively, until the error was eliminated.

First scenario (single source, single receiver): In this scenario we consider S2 and R2 in figure 11. Consider the sequence of events in the figure.

Packet duplication and *register* looping occur in the above scenario. A similar scenario occurs when R2 joins first then S2 starts sending to the group.

Suggested fixes The required behavior is to send a –triggered and periodic– source-specific *prune* off of the shared tree, if a router has source-specific state for registering and shared tree state for the same group (regardless of the incoming interface settings).

Figure 10: The Join/Prune scenario under study

Second scenario (single sender, two receivers): We assume the implementation of the above fixes to the simulator, then consider the sequence of events in figure 12. This scenario exhibits packet duplication and *register* looping.

Suggested fix The problem arises because the packets are forwarded back on the originating LAN, and treated as if they were new packets originated by the directly connected source. The following rule solves this problem for the given scenario:

- A router receiving *join* message must **NOT** add an interface on the same subnet as a source S, for any source specific entry for S associated with same group.

Third scenario (single source, single receiver with message loss): Considering the scenario in figure 13.

The source specific *prune* sent from A to C–when A having a shared tree state, creates the source specific entry for registering– is lost.

Packet duplication and *register* looping problems are experienced in this scenario. The problem persists until a periodic *Join/Prune* message is successfully sent upstream.

Suggested fix To be robust to (at least) one message loss, we suggest the following rule for packet forwarding:

- A router must **NOT** forward a packet onto the subnet from which the packet was originated. This is achieved by performing a check on the source and the outgoing interface before building a source

Figure 11: The first *register* scenario under study

specific state or before forwarding a packet[16].

5 Related work

The related work falls mainly in the field of protocol verification. Most of the literature on multicast protocol design addresses architecture, specification, and comparisons between different protocols. We are not aware of any other work to develop systematic methods for testing multiparty protocol robustness. In addition, some concepts of STRESS were inspired by VLSI chip testing.

There is a large body of literature dealing with verification of communication protocols. Protocol verification typically addresses *safety*, *liveness*, and *responsiveness* properties [22]. Safety properties include freedom from deadlocks, assertion violations, improper terminations and unspecified receptions. Liveness properties include detection of acceptance cycles and absence of non-progress cycles. While responsiveness properties include timeliness and fault tolerance. Most protocol verification systems, including STRESS, aim to detect violations of (part of) these protocol properties.

In general, the two main approaches for protocol verification are theorem proving and reachability analysis (or model checking) [23, 24]. Theorem proving systems define a set of axioms and construct relations on these axioms. Desirable properties of the protocol are then proven mathematically. Theorem proving includes *model-based* formalisms (such as Z [25], and Vienna Development Method (VDM) [26]) and *logic-based* formalisms including first order logic (such as Nqthm [27]) and higher order logic (such as Prototype Verification System (PVS) [28]). Formal verification has been applied to TCP and T/TCP [29], but for multiparty protocols, theorem proving systems are likely to be even more complex and perhaps intractable.

Reachability analysis algorithms [1, 2] try to generate and inspect all the protocol states that are reachable from given initial state(s). Such algorithms suffer from the 'state space explosion' problem, especially in complex systems as are multiparty protocols. To circumvent this problem, state reduction and controlled partial search techniques [30, 31] could be used. These techniques focus only on parts of the state space and may use probabilistic [32], random [33] or guided searches [34]. The SPIN tool [35] uses the *supertrace* algorithm that is actually a random controlled partial search. STRESS has similarities with guided controlled partial searches. However, STRESS explores protocol states based on the *representative scenarios*, and does not use a cost function as does guided search.

[16] Most implementations create a cache for forwarding packets. This check can be done only once when creating the cache, and is not done per packet.

This is different than the 'incoming interface' check stated by the current specification. In the specific case discussed here, the looping multicast packets will match on the incoming interface (the LAN) for the source-specific entry.

Figure 12: The second *register* scenario used for the study

There is an analogy between STRESS and VLSI systematic design for testability [36] using Built-In-Self-Test (BIST) [37]. BIST provides a systematic technique for chip testing synthesis. This technique can be used to detect faults due to single-stuck-line, while STRESS can be used to detect errors due to single packet loss. BIST uses a test pattern generator (TPG) to produce the input patterns applied to the circuit under test. Conceptually, this resembles our use of the scenario generator. The test patterns are chosen to maximize fault coverage with a minimum number of inputs. Similarly, the scenario filter chooses the representative scenarios to maximize protocol coverage with a minimum number of scenarios. Moreover, BIST uses a 'response monitor circuit' to monitor and detect error signals. This is analogous to our use of tracing and error detection modules.

The expected output for VLSI chip testing is 'fault coverage vs. test length' curve. Although this is similar to 'protocol coverage vs. scenarios' statistics, we add the coverage of possible interaction scenarios as another output dimension.

6 Summary and Future Work

The goals of our method are to simplify and systematize robustness analysis of multiparty protocols. This paper presented our initial attempts to achieve these goals in the context of one multicast routing protocol. We used scenario generation, simulation tracing, and output analysis to obtain a set of error-prone scenarios. In particular, we described several techniques:

- To circumvent the 'state explosion problem', we introduced the notion of *representative scenarios.* We obtained these scenarios for the multicast host model using a *scenario filter*, which excluded redundant and irrelevant scenarios based on practical assumptions.
- We identified two *representative topologies* based on the *equivalence* relationship established for the protocol under study, PIM-SM. The equivalence definition suggested that extending the simulated topologies would not reveal additional errors in the protocol.
- To capture robustness characteristics, we studied the protocol behavior in the presence of single packet loss. A *LAN* module accounted for the *selective loss* cases experienced by multicast messages. We hope to use a similar *logical LAN* module, to model loss and delay parameters of the underlying multicast distribution trees and thereby extend our method to higher-level multiparty protocols.
- To reduce the complexity of our analyses, we used *subsetting* of protocol functions, states and messages. This allowed abstracting out some protocol details while retaining and focusing on others.

Figure 13: The third *register* scenario used for the study

- The definition of *error conditions*, in terms of end-point errors (such as data loss or duplication), enabled the output analyzer to capture *faulty scenarios* and isolate protocol traces in the recent history of the errors.

- Finally, we showed that a large portion of the protocol state space could be covered by simulating a few *representative scenarios*. With the aid of STRESS, we were able to discover several protocol design errors in PIM-SM, and suggest solutions to these errors.

This paper was the first attempt to develop and apply these methods. We are encouraged by our success in identifying protocol errors using these methods, and hope that we have similar results as we move on to investigate other multicast routing and end-to-end multiparty protocols.

Future directions for this research include:

- Further applying STRESS to multicast routing to investigate a wider range of protocol functions:
  - timers and timed actions, to complement the triggered actions investigated in this study.
  - heterogeneous topologies including asymmetric and uni-directional links, such as satellite links.
  - other multicast routing protocols, such as DVMRP, PIM-DM and hierarchical PIM [38].
  - interoperability between routing protocols. Examples include interaction between unicast and multicast routing, and the interoperability between multicast routing protocols.

- Generalizing the method and extending it to apply to multiparty protocols. Examples of such protocols include reliable and real-time multicast transport and session management protocols, such as SRM, RTCP, and sdr. To achieve this, the multicast distribution tree may be viewed as a *logical LAN*, with various selective loss and delay models.

- Applying to real implementation conformance testing [39] through an emulation interface. This facilitates driving the test using representative scenarios.

- Deriving behavioral assertion checks [40] based on the STRESS method. Assertions can be used in network management and self-diagnosing protocols.

7 Acknowledgements

We would like to thank our colleagues at USC/ISI; George Eddy, John Hiedeman, Kanna Kumar, and Pavlin Radoslavov for their useful comments on early versions of the paper.

References

[1] F. Lin, P. Chu, and M. Liu. Protocol Verification using Reachability Analysis. *Computer Communication Review, Vol. 17, No. 5*, 1987.

[2] F. Lin, P. Chu, and M. Liu. Protocol Verification using Reachability Analysis: the state explosion problem and relief strategies. *Proceedings of the ACM SIGCOMM'87*, 1987.

[3] D. Waitzman S. Deering, C. Partridge. Distance Vector Multicast Routing Protocol, November 1988. RFC1075.

[4] J. Moy. Multicast Extension to OSPF. *Internet Draft*, September 1992.

[5] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification. *Proposed Experimental RFC. URL http://netweb.usc.edu/pim/pimdm/PIM-DM.{txt,ps}.gz*, September 1996.

[6] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core Based Trees. In *Proceedings of the ACM SIGCOMM*, San Francisco, 1993.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Motivation and Architecture. *Proposed Experimental RFC. URL http://netweb.usc.edu/pim/pimsm/PIM-Arch.{txt,ps}.gz*, October 1996.

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, November 1996.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *RFC 1889*, January 1996.

[10] S. McCanne. A Distributed Whiteboard for Network Conferencing. *U.C. Berkeley Computer Science project*, May 1992.

[11] V. Jacobson and S. McCanne. vat - LBNL Audio Conferencing Tool. *URL http://www-nrg.ee.lbl.gov/vat*, 1993.

[12] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. *ACM Multimedia 1995*, November 1995.

[13] M. Handley. NTE - The UCL Network Text Editor. *URL http://www-mice-nsc.cs.ucl.ac.uk/mice-nsc/tools/nt-help:about.html*, 1996.

[14] M. Handley. The sdr Session Directory: An Mbone Conference Scheduling and Booking System. *URL http://ugwww.ed.ac.uk/mice/archive/sdr.html*, 1996.

[15] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification. *RFC 2117. URL http://netweb.usc.edu/pim/pimsm/PIM-SMv2-Exp-RFC.{txt,ps}.gz*, March 1997.

[16] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing. *Submitted to IEEE/ACM Transactions on Networking. URL http://www.usc.edu/dept/cs/technical_reports.html*, May 1997.

[17] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik, Vol. 1*, 1959.

[18] S. McCanne and S. Floyd. NS: Network Simulator. *URL http://www-nrg.ee.lbl.gov/ns*, 1995.

[19] J. Ousterhout. Tcl and the Tk Toolkit. *Addison Wesley*, 1994.

[20] D. Wetherall and C. Lindblad. Extending Tcl for Dynamic Object-Oriented Programming. *Proceedings of the Tcl/Tk Workshop 95, Toronto, Ontario*, July 1995.

[21] A. Helmy. Protocol Independent Multicast-Sparse Mode (PIM-SM): Implementation Document. *Internet Draft. URL http://www.usc.edu/dept/cs/technical_reports.html*, January 1997.

[22] K. Saleh, I. Ahmed, K. Al-Saqabi, and A. Agarwal. A recovery approach to the design of stabilizing communication protocols. *Journal of Computer Communication, Vol. 18, No. 4*, pages 276–287, April 1995.

[23] E. Clarke and J. Wing. Formal Methods: State of the Art and Future Directions. *ACM Workshop on Strategic Directions in Computing Research, Vol. 28, No. 4*, pages 626–643, December 1996.

[24] A. Helmy. A Survey on Kernel Specification and Verification. *Technical Report 97-654 of the Computer Science Department, University of Southern California. URL http://www.usc.edu/dept/cs/technical_reports.html.*

[25] J. Spivey. Understanding Z: a Specification Language and its Formal Semantics. *Cambridge University Press*, 1988.

[26] C. Jones. Systematic Software Development using VDM. *Prentice-Hall Int'l*, 1990.

[27] R. Boyer and J. Moore. A Computational Logic Handbook. *Academic Press, Boston*, 1988.

[28] S. Owre, J. Rushby, N. Shanker, and F. Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, pages 107–125, February 1995.

[29] M. Smith. Formal Verification of Communication Protocols. *FORTE/PSTV'96 Conference*, October 1996.

[30] D. Probst. Using partial-order semantics to avoid the state explosion problem in asynchronous systems. *Proc. 2nd Workshop on Computer-Aided Verification, Springer Verlag, New York*, 1990.

[31] P. Godefroid. Using partial orders to improve automatic verification methods. *Proc. 2nd Workshop on Computer-Aided Verification, Springer Verlag, New York*, 1990.

[32] N. Maxemchuck and K. Sabnani. Probabilistic verification of communication protocols. *Proc. 7th IFIP WG 6.1 Int. Workshop on Protocol Specification, Testing, and Verification, North-Holland Publ., Amsterdam*, 1987.

[33] C. West. Protocol Validation by Random State Exploration. *Proc. 6th IFIP WG 6.1 Int. Workshop on Protocol Specification, Testing, and Verification, North-Holland Publ., Amsterdam*, 1986.

[34] J. Pageot and C. Jard. Experience in guiding simulation. *Proc. VIII-th Workshop on Protocol Specification, Testing, and Verification, Atlantic City, 1988, North-Holland Publ., Amsterdam*, 1988.

[35] G. Holzmann. Design and Validation of Computer Protocols. *AT&T Bell Labs., Prentice Hall*, 1991.

[36] B. Murray and J. Hayes. Testing ICs: Getting to the Core of the Problem. *IEEE Computer Magazine*, pages 32–38, November 1996.

[37] B. Konemann, B. Bennetts, N. Jarwala, and B. Nadeau-Dostie. Built-In Self-Test: Assuring System Integrity. *IEEE Computer Magazine*, pages 39–45, November 1996.

[38] S. Deering, B. Fenner, D. Estrin, A. Helmy, D. Farinacci, L. Wei, M. Handley, V. Jacobson, and D. Thaler. Hierarchical PIM-SM Architecture for Inter-Domain Multicast Routing. *Internet Draft. URL http://netweb.usc.edu/ahelmy/interdomain-multicast/interdom-1-19.ps*, December 1995.

[39] D. Rayner. OSI conformance testing. *Computer Networks and ISDN Systems, Special issue on Conformance Testing, Vol. 14, No. 1*, pages 79–98, 1987.

[40] S. Perl. Performance Assertion Checking. *Ph.D. Thesis, MIT*, September 1992.