Simulation-based 'STRESS' Testing Case Study: 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 *

Abstract

In this work, we propose a method for using simulation to analyze the robustness of multiparty (multicast-based) protocols in a systematic fashion. We call our method **S**ystematic **T**esting of **R**obustness by **E**xamination of **S**elected **S**cenarios (STRESS). *STRESS aims to cut the time and effort needed to explore 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 aim to offer design tools similar to those used in CAD and VLSI design, and demonstrate how effective systematic simulation can be in studying protocol robustness.*

1 Introduction

Multiparty protocols support applications ranging from conferencing to data dissemination and network games. Designing wide-area multiparty protocols is becoming more complex with the growth of the Internet and the introduction of new service models. Anticipating errors in such protocols often requires extensive simulation and testing, and sometimes unexpected cases are not observed until deployment.

In this paper, we describe a simulation framework (referred to as STRESS) supported by a set of tools, designed for studying protocol behavior in the context of pathological cases. Some of the general concepts for STRESS draw from simulation-based verification techniques and reachability analysis [12]. 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.

We apply these techniques to support the design, analysis, and testing of multiparty protocols; specifically, multicast routing. As a case study, we apply our method to the Protocol Independent Multicast-Sparse Mode (PIM-SM) [5]. Our study revealed several pathological errors in PIM-SM, and evaluated solutions to eliminate these errors.

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 Method Overview

The *robustness* of a protocol is its ability to respond correctly to failures and packet loss. The goal of *STRESS* 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 iteratively evaluate design trade-offs, analyze behavior, and test implementations of the protocol.

As shown in figure 1, our basic approach consists of three stages: *scenario generation* (pre-processing), *tracing* (simulation), and *output analysis* (post-processing). These stages are explained in the rest of this section.

2.1 Scenario Generation

Scenarios describe the simulation environment including *routed topology*, *host scenarios* and *loss scenarios*.

Routed topology The routed topology is the network infrastructure upon which the protocol operates; e.g. nodes,

*This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0054.

Figure 1. *STRESS* **method block diagram**

As a component of the routed topology, unicast routing may be a common source of error. Unicast routing inconsistencies may be either (a) transient, or (b) long lived, which may be due to a multicast region spanning more than a unicast routing AS. The study of case (a) is convergence analysis, which has been addressed elsewhere [6]. We are interested in case (b). We add an inconsistent unicast routing component to force the multicast routing protocol into states encountered in such pathology, and analyze those states.

Host scenarios Host scenarios are combinations of possible host actions. For multicast routing, these actions include joining, leaving, or sending packets to groups. For large numbers of hosts and groups it is prohibitively costly to explore all possible combinations exhaustively.

The heuristics used herein do not guarantee that all faulty scenarios for a protocol are covered. Our more practical and achievable objective is to study multicast protocol behavior for scenarios that include the primary host events. For these scenarios, we generate all possible message loss cases and extract the faulty scenarios automatically.

To illustrate, we choose a simple scenario that has one source and two receivers 'R1' and 'R2' for the same group. We estimate all possible combinations of our host model, then 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 practical and protocol contraints. For example, the above scenario has five possible host events: source sending to a group , receiver joining a group ('J1' and 'J2' for 'R1' and 'R2', respectively), and receiver leaving a group ('L1' and 'L2').

For all possible permutations, there exists $5! = 120$ scenarios, considering that each host event occurs once. Then we apply protocol constraints, (e.g. *a receiver cannot leave before it joins the group*), to reduce the number of possible combinations to $5!/(2! \times 2!) = 30$ scenarios. Further, as a practical constraint, we assume that (*the source sends packets throughout the simulation*), to reduce the number of possible scenarios to $30/5 = 6$ scenarios, as follows:

```
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
```

Loss and Failures The input to the 'loss & failures' substage (shown in figure 1) is obtained from initial traces of simulations without protocol message loss. These traces guide further simulations to cover all possible protocol message loss scenarios.

The loss and failure scenarios include loss of state in routers (e.g. due to crashes), or loss of packets. Packet loss may occur due to congestion or failures. We classify these events as simply 'packet loss', and create exhaustive loss scenarios to capture all the possible protocol transitions and pathologies due to packet loss.

We consider single fault models; those that address the occurrence of a single fault per scenario. In particular, the loss of a single protocol message by any of the intended receivers.

For most multicast protocols, hop-by-hop messages are multicast on multi-access network (LANs), and may experience selective loss; i.e. may be received by some nodes but not others. We use the term LAN to designate a connected network with respect to IP; this includes shared media (such as Ethernet, or FDDI), hubs, switches, and other network devices. The likelihood of selective loss is increased when LANs contain multiple 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.

2.2 Simulation and Tracing

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

Simulation One desirable approach for simulating complex protocols, is to include detailed mechanisms of parts

Figure 2. PIM-SM rendezvous scenario

As shown in figure 2, when a receiver's local router (*A*) discovers it has local receivers, it sends *join* message toward a Rendezvous-Point (RP). The *join* messages are multicast hop-by-hop. Each router along the path toward the RP builds a *route entry* and sends the *join* messages on toward the RP. A route entry is the state held in a router to maintain the shared distribution tree and includes the source and group addresses, the interface from which packets are accepted (*incoming interface*), and the list of interfaces to which packets are sent (*outgoing list*). Upon arriving at a router, a multicast packet is forwarded according to the route entry.

Figure 3. The equivalent topologies

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 3-router LAN topology for prunes, w.r.t. single message loss scenarios.*

To justify our assumption we first prove that a *4-router* LAN topology is reducible to a *3-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.

Next, we examine the *3-router* LAN topology. In figure 3, topology 'I', assume that *A* and *B* are downstream routers, and *C* is an upstream router.

- In figure 3, 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 selective loss pattern for the *prune* message sent by *B* (see left table).

A	C	
0	0	
0	1	
1	0	← error
1	1	

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	

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

- In figure 3, topology 'II', we add another downstream router *D*. The selective loss pattern table is given above.

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*. With the addition of an upstream router (figure 3, topology 'III'), no added error cases are encountered.

Similarly, in [10] we show that the *N+1-router* LAN topology is reducible to the *N* case; where $N \geq 3$.

From the above we see that by simulating the *3-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 $N > 3$.

In [10] we establish similar equivalence for the *Join* and *Assert*, and conclude that a *4-router* LAN topology is an equivalent topology for PIM-SM.

Figure 4. Topology used

For our case study, we use a *4-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 4.

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 4. For the unicast routing protocol we use a centralized version of Dijkstra's SPF algorithm.

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 4, topology 1.

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. The source is modeled as a constant-bit rate (CBR) stream with fixed packet size. The 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[1].

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.

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

[1]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.

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 1. As we will show in section 4, 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), node, type of message, and time. Every data packet is assigned a unique sequence number.

For example, the trace '**R2 Node A Rcv 7 t 190**' means that receiver **R2** in node **A** received data packet **7** at **190ms**.

Subsetting As an example of state subsetting, we only consider shared group states, but not 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 4, topology 1). This is an example of *message subsetting*.

When studying *registers*, *joins* and *prunes* we consider topology 2 in figure 4 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*.

3.4 Applying the Method

This section describes the simulator and gives an illustrative example, to show how STRESS may be used to identify and analyze protocol errors.

The Simulation Framework We have implemented an initial version of the STRESS method in the Network Simulator 'NS'[2]. NS is an event-driven packet-level simulator controlled and configured via OTcl.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, based on 'pimd' [8]. 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.

Figure 5 depicts the network and protocol simulation modules, explained next.

[2]For the simulator see http://catarina.usc.edu/vint.

Figure 6. Packet traces

topology 1. Traces in figure 6 give partial history of the errors found. The first error (i.e. the packet duplication) has the host event 'J2' as the closest host event in its history at time 200ms. This transient error is caused by parallel paths to the RP, and 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.

The above end-point errors are considered transient errors, but not design errors.

4 Results

This section describes the protocol design errors revealed for PIM-SM under STRESS, followed by an evaluation of the protocol coverage achieved by the study. For a detailed discussion of the protocol errors and fixes see [10].

Unlike our example above, we are only interested in non-transient errors. For this, we have modified the error conditions to not consider single duplication or loss.

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 4, 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 4, 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 4, 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.

For overall protocol coverage we considered two metrics: a) scenario coverage by investigating the *selective loss* scenarios, and b) code coverage using *representative* scenarios simulations.

Scenarios covered The initial number of simulated scenarios *without* protocol message loss was 12; 6 (in section 2.1), over 2 topologies (in section 3.3).

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:

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

Term	Meaning
Topos	Topologies
Reps	Representative Scenarios
Msgs	Messages under study
LinkMsgs	No. messages traversing the link
LinkRtrs	No. routers connected to the link

For each topology, this formula gives the number of scenarios automatically generated after the first (loss free) simulation run, during which the number of messages and links (traversed by these messages) is counted.

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, with a total of 352 scenarios with loss. For the second topology, there were 296 scenarios with loss.

Protocol code coverage A large portion of the multicast support code in NS was annotated automatically to provide code tracing. Out of 91 procedures (procs), following are the procedures covered by the representative scenarios:

Topology	Procs covered	%ge
Topology1	79	86.8%
Topology2	80	87.9%
Total	84	92.3%

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.

5 Related work

The related work falls mainly in the field of protocol verification. 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* (e.g. deadlock freedom), *liveness*

(e.g. livelock freedom), and *responsiveness* (e.g. timeliness) properties. Most protocol verification systems aim to detect violations of these properties.

In general, the two main approaches for protocol verification are theorem proving and reachability analysis (or model checking) [3]. 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 (e.g. VDM [11]) and *logic-based* formalisms (e.g. Nqthm [2]). For multiparty protocols, however, theorem proving systems are likely to be even more complex and perhaps intractable.

Reachability analysis algorithms 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 [7] could be used. STRESS has similarities with controlled partial searches, but explores protocol states based on the representative scenarios.

There is an analogy between STRESS and VLSI systematic design for testability using Built-In-Self-Test (BIST) [1]. BIST provides a systematic technique for chip testing synthesis, and can be used to detect faults due to single-stuck-line.

BIST uses a 'test generator' to produce the input patterns applied to the circuit under test, and 'response monitor circuit' to monitor and detect error signals. The test patterns are chosen to maximize fault coverage with a minimum number of inputs. Conceptually, this resembles the STRESS framework. However, VLSI testing is performed on a given circuit, whereas network protocol robustness must be established over arbitrary and time-varying topologies.

6 Summary and Future Work

The goals of our method are to systematize and provide tools for 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: a) *representative scenarios* and *equivalent topologies*, to circumvent the 'state explosion problem', b) *selective loss* over LANs, to capture robustness characteristics, c) *subsetting*, to reduce the complexity of our analysis, d) the definition of *error conditions*, to enable automatic capture of *faulty scenarios*, and e) *scenario and code coverage*, to evaluate the space covered by the simulations.

Using STRESS, we were able to discover several errors in PIM-SM, and suggest solutions to these errors.

Future directions for this research include: a) Developing algorithms for automatic scenario generation, that replace the heuristics used in this study[3], b) Investigating a richer set of scenarios, including timers, heterogeneous topologies with asymmetric and uni-directional links, other protocols (e.g. PIM-DM [4]) and multicast interoperability, c) Extending the method to apply to end-to-end multiparty protocols. To achieve this, the multicast distribution tree may be viewed as a *logical LAN*, with various selective loss and delay models, and d) Applying conformance testing to real implementation through an emulation interface.

References

[1] M. Abramovici, M. Breuer, and A. Friedman. Digital Systems Testing and Testable Design. *AT & T Labs*, 1990.

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

[3] 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*, Dec. 1996.

[4] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei. Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification. *Experimental RFC*, Sept. 1996.

[5] 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*, Mar. 1997.

[6] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing. *Sub. IEEE/ACM Trans. on Networking*, May 1997.

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

[8] A. Helmy. Protocol Independent Multicast (PIM-SM): Implementation Document. *Internet Draft*, Jan. 1997.

[9] A. Helmy. Systematic Testing of Multicast Protocol Robustness. *USC-CS-TR 98-663, www.usc.edu/dept/cs*, Dec. 1997.

[10] A. Helmy and D. Estrin. Simulation-based 'STRESS' Testing Case Study: A Multicast Routing Protocol. *USC-CS-TR 98-674, www.usc.edu/dept/cs*, Mar. 1998.

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

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

[3] We are currently investigating semi-formal approaches, based on well-established VLSI testing techniques [1], and extending them to synthesize test topologies and scenarios automatically (see [9]).