Fault-oriented Test Generation for Multicast Routing

Ahmed Helmy, Deborah Estrin, Sandeep Gupta
University of Southern California
Los Angeles, CA 90089
email:{ahelmy,estrin}@usc.edu, sandeep@boole.usc.edu *

March 26, 1998

Abstract

The unprecedented growth of the Internet and the introduction of new network services, such as multicast, has lead to the increased complexity of network protocols and protocol interaction. Multicast protocols support a wide range of multipoint applications ranging from teleconferencing to network games. Unlike traditional point to point protocols, multipoint communication involves multiple senders and receivers, increasing the number of protocol states, and complicating the task of evaluating the behavior and robustness of the protocols and supported applications.

In addition, the heterogeneity of network components and technologies has introduced new failure modes that have not been considered traditionally in the design of multicast protocols; such as unicast routing anomalies and selective loss over LANs. The presence of these failures exacerbates the design and testing problems of multicast protocols, due to the esoteric interaction between the different layers in the protocol stack. To date, little effort has been exerted to formulate practical methods and tools that aid in the systematic testing of these protocols.

In this paper we present a new algorithm for automatic test generation for multicast routing. We target protocol robustness in specific, and do not attempt to verify other properties in this paper. Our algorithm processes a finite state machine (FSM) model of the protocol and uses a mix of forward and backward search techniques to generate the tests. The output tests include a set of topologies, protocol events and network failures, that lead to violation of protocol correctness and behavioral requirements. We apply our method to a real multicast routing protocol, PIM-DM –which has been deployed in parts of the Internet–, and investigate its behavior in the presence of selective packet loss on LANs and router crashes.

1 Introduction

Network protocol errors are often detected by application failure or performance degradation. Such errors are hardest to diagnose when the behavior is unexpected or unfamiliar. Even if a protocol is proven to be correct in isolation, its behavior may be unpredictable in an operational network, where interaction with other protocols and the presence of failures may affect its operation.

The complexity of network protocols is increased with the exponential growth of the Internet, and the introduction of new services. In particular, the advent of IP multicast and the MBone enabled applications ranging from multi-player games to distance learning and teleconferencing, among others.

In addition, researchers are observing new and obscure, yet all too frequent, failure modes over the internets; such as routing anomalies [1, 2], and selective loss over LANs [3]. Such failures are becoming more frequent, mainly due to the increased heterogeneity of technologies and configuration of various network components.

Many researchers have developed protocol verification methods, to ensure that certain properties of a protocol hold; properties like freedom from deadlocks or unspecified receptions. Much of this work, however, was based on abstract mathematical models, with assumptions about the network conditions (such as FIFO queues or not considering network component failures), that may not always hold in today's Internet, and hence may become invalid.

To provide an effective solution to these problems, we propose a new method for automatic test generation. We refer to our method as the fault-oriented test generation (FOTG). It is targeted towards the study of protocol robustness in the presence of packet loss and network failures.

*Helmy and Estrin were supported by the Defense Advanced Research Projects Agency (DARPA) under Contract 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.

Our approach borrows from well-established chip testing technologies. We adopt concepts –such as 'fault-oriented pattern generation', and 'forward and backward implication'–, expand and apply them to network protocols.

The algorithm used in our method utilizes a mix of forward and backward search techniques. Starting from a given fault, the necessary topology and event sequences are established, that drive the protocol into error states. For illustration, and as a case study, we apply FOTG to a multicast routing protocol deployed as an intra-domain routing protocol in parts of the Internet; PIM-DM [4].

The rest of this paper is organized as follows. Section 1.1 gives an overview of multicast. The related work on protocol design and testing is discussed in section 2. Section 3 provides an overview of test generation and presents the system model and definitions. Section 4 describes our algorithm in detail, and how it can be applied to multicast routing, along with the results of our case study. We conclude by a summary and a discussion of future directions in section 5.

1.1 Brief Overview of Multicast

Multicast protocols are the class of protocols that support group communication. A multicast group may involve multiple receivers and one or more senders. In this paper, we address multicast protocols for the Internet, based on the IP multicast model. These protocols include multicast routing protocols (e.g. DVMRP [5], MOSPF [6], PIM-DM [4], CBT [7], and PIM-SM [8]), multicast transport protocols (e.g. SRM [9], RTP, and RTCP [10]) and multiparty applications (e.g. WB [11], vat [12], vic [13], nte [14], and sdr [15]). This study focuses on multicast routing protocols, which deliver packets efficiently to group members by establishing distribution trees. Figure 1 shows a very simple example of a source S sending to a group of receivers R_i.

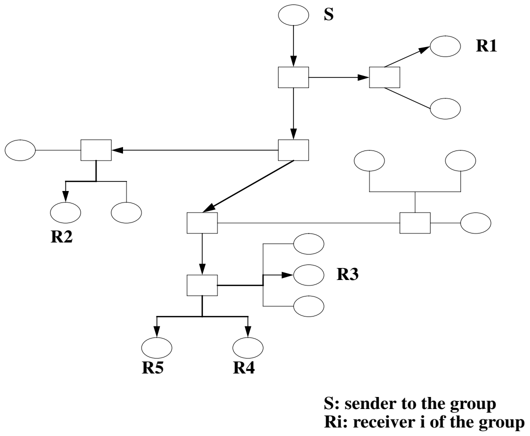

Figure 1: Establishing multicast delivery tree

Multicast distribution trees may be established by either broadcast-and-prune or explicit join protocols [3]. 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 receiving multicast packets, a router forwards the packets according to the routing state.

We are particularly interested in multicast routing protocols, because they are vulnerable to failure modes, such as selective loss, that have not been traditionally studied in the area of protocol design.

For most multicast protocols, when routers are connected via a multi-access network (or LAN)[1], hop-by-hop messages are multicast onto 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, end-to-end multicast 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.

[1]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.

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 are even more likely to experience selective loss among receivers.

We are also interested in studying the multicast protocol behavior in the presence of other network failures, such as router crashes (considered in this paper), and unicast routing anomalies (considered in the future work).

2 Related Work

The related work falls mainly in the field of protocol verification. In addition, some concepts of our work were inspired by VLSI chip testing. 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 automatic test generation for multicast protocol robustness.

There is a large body of literature dealing with verification of communication protocols. Protocol verification typically addresses well-defined properties, such as *safety*, *liveness*, and *responsiveness* properties [16]. 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 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) [17, 18]. 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 [19], and Vienna Development Method (VDM) [20]) and *logic-based* formalisms including first order logic (such as Nqthm [21]) and higher order logic (such as Prototype Verification System (PVS) [22]). An attempt to apply formal verification to TCP and T/TCP has been given in [23]. In general, however, the number of axioms and relations in theorem proving systems grows with the complexity of the protocol. We believe that these systems will be even more complex, and perhaps intractable, for multicast protocols. Moreover, these systems work with abstract specification of the protocol, and hence tend to abstract out some network dynamics (such as selective loss or unicast routing inconsistencies) or failures that may cause problems we are addressing in this study.

Reachability analysis algorithms [24, 25], on the other hand, 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 for complex protocols. To circumvent this problem, state reduction and controlled partial search techniques [26, 27] could be used. These techniques focus only on parts of the state space and may use probabilistic [28], random [29] or guided searches [30]. Reduced reachability analysis has been used in the verification of cache coherence protocols [31], using a global FSM (finite state machine) model. We adopt a similar FSM model and extend it for our approach in this study.

A simulation-based STRESS testing method was proposed in [3]. Although this method proposes heuristics and topological equivalence to reduce the number of simulated scenarios, it does not provide automatic generation of topologies and scenarios. Our work in this paper may be integrated with the STRESS framework as part of the scenario generation phase.

Another related field is VLSI chip testing. Chip testing uses a set of well-established approaches to generate test vector patterns, generally for detecting physical defects in the VLSI fabrication process. Common test vector generation methods detect single-stuck faults; where the value of a line in the circuit is always at logic '1' or '0'. Test vectors are generated based on a model of the circuit and a given fault model [32].

One well-known testing process for stuck-at faults is the fault-oriented process. In this process, the two fundamental steps in generating a test vector are: a) activating the fault, and b) propagating the resulting error to an observable output. Activating a fault involves a line justification step; i.e., setting circuit input values to cause a line in the circuit to have a specific value.

Line justification or error propagation usually involve a search procedure with a backtracking strategy to resolve or undo contradiction in the assignment of line and input values. These line assignments sometimes determine or imply other line assignments. The process of computing the line values to be consistent with previously determined values is referred to as *implication*[2]. Forward implication involves implying values of lines from the fault toward the output, while backward implication involves implying values of lines from the fault toward the circuit input.

[2] For chip testing, *implication* often refers to a unique assignment. In our usage of the term, however, we allow multiple possible values, in which case a search procedure is used to investigate the possibilities.

We adopt some implication concepts for our method, and transform them to the network protocol domain. We note that chip testing is performed for a given circuit, while a protocol must work over arbitrary and time varying topologies, adding another dimension to our problem.

3 Test Generation Overview

The input to our method is the specification of a protocol, its correctness requirements, and a definition of its robustness. In general, protocol robustness is the ability to respond correctly in the face of network failures and packet loss. Usually robustness is defined in terms of network dynamics or fault models. A fault model represents various component faults; such as packet loss, corruption, re-ordering, or machine crashes. The desired output is a set of test-suites that stress the protocol mechanisms according to the robustness criteria.

The core contribution of our work lies in the development of systematic test generation algorithms for protocol robustness.

In general, there are two approaches for test generation (TG); random TG (RTG) and deterministic TG [32]. RTG involves only the generation of random test patterns (see section 3.3 for the definition of test patterns), and hence is simple. However, a large set of test patterns is needed to achieve a high measure of error coverage, and even then determining the test quality may be expensive. Also, the cost of running long test sequences may be high. RTG generally does not take into account the function or the structure of the protocol under test, and does not attempt to minimize the test length.

Deterministic TG, on the other hand, produces tests based on a model of the protocol. Hence, it may be more expensive than RTG. However, the knowledge built into the protocol model enables the production of shorter and higher-quality test sequences. Deterministic TG can be manual or automatic. In this study we focus on automatic deterministic TG (ATG).

ATG can be: a) fault-independent, or b) fault-oriented. Fault-independent TG (FITG) works without targeting individual faults as defined by the fault model. Such an approach may employ a forward search technique to inspect the protocol state space (or an equivalent subset thereof), after integrating the fault into the protocol model. In this sense, it may be considered a variant of reachability analysis, and may use equivalence relations to reduce the state space investigated. In general, FITG may be used to automatically construct sequences of events that lead to error states over a given topology.

In contrast, fault-oriented tests are generated for specified faults. Fault-oriented test generation (FOTG) starts from the fault (e.g. a lost message) and synthesizes the necessary topology to trigger that message. It then uses a mix of forward and backward searches to construct sequences of events leading to protocol error. In general, FOTG requires more information in the protocol model for the topology synthesis and the backward search. In section 4, we will discuss in greater detail the protocol model used and apply it to a multicast protocol; PIM-DM.

In the remainder of this section, we describe the system model, present some definitions, and give an overview of the case study protocol, PIM-DM.

3.1 System Model and Definition

The system model consists of the network elements, topology elements and the fault model.

Elements of the network The network consists of links and nodes; routers and hosts. A link may be point-to-point or multi-access (i.e. LAN). In this study we assume bi-directional symmetric links. A node runs a set of network protocols; unicast and multicast routing. We assume the existence of a MAC layer protocol to resolve media access and collision issues, but we do not model such protocol. A host runs end-to-end protocols or applications.

Elements of the topology In this document we consider only local topology [N-router LAN] modeled at the network level (i.e. connecting hubs, switches, bridges and other data-link-layer devices are abstracted out). The boundary of our topology is the multicast routing domain, which contains only a single multicast routing protocol. However, the topology may span multiple unicast routing domains or Autonomous Systems (ASs).

The fault model We distinguish between the terms *error* and *fault*. An *error* is the failure of a protocol to meet its design requirement. For example, duplication in packet delivery is an error for multicast routing. A *fault*, on the other hand, is a low level (e.g. physical layer) anomalous behavior, that may affect the behavior of the protocol under test, and include, for example packet loss or unicast route flapping, among others. Note that a fault may not necessarily be an error for the low level protocol.

The fault model may include:

- Loss of packets due to queue congestion, overflow, link failures, or packet corruption. We assume that the packets are either delivered correctly, or are dropped; i.e. packet corruption is discovered using checksum or other error detection codes.
- Loss of state, such as multicast and/or unicast routing tables due to failure of the routing protocol, crashes, or insufficient memory resources. The duration of this loss varies with the nature of the failure. For example, for glitches, the loss may last for portions of a second, whereas for momentary machine crashes, the loss may last for minutes.
- The delay model: Delays in the network may be due to transmission, propagation, or queuing delays. We assume that the processing delays are negligible w.r.t. the time granularity the analysis is addressing. Sometimes delay problems can be translated into sequencing problems, as we will show by example in section 4.6.
- Unicast routing anomalies, such as route inconsistencies, oscillations or flapping.

Usually, a fault model is defined in conjunction with the robustness criteria for the protocol under study (in our case PIM). A fault model may include a single fault or multiple faults. In our study, we adopt a single-fault model, where only a single fault may occur during a scenario or a test sequence. A design requirement for PIM is to be robust to single protocol message loss [3].

We also study the behavior of the protocol in the presence of momentary loss of state.

3.2 Test Sequence Definition

Given two sequences $T =< e_1, e_2, \ldots, e_n >$ where e_i is an event, and $T' =< e_1, e_2, \ldots, e_k, f, e_j, \ldots, a_n >$, where f is a fault[4]. Let $P(q,T)$ be the sequence of states and stimuli of protocol P under test T starting from the initial state q. According to one of the following definitions T' may be said to be a test sequence if:

1. $P(q,T) \neq P(q,T')$. This means that the behavior of the system in the presence of the fault is different than that without the fault. Note that this definition may include sequences that (including and excluding the fault) produce the same correct final states, but with different transient behavior, or
2. Final $P(q,T) \neq P(q,T')$; i.e. the stable state (after the occurrence of the fault) is different for the two outputs. This definition ignores transient behavior, but may include sequences that (including and excluding the fault) produce different correct final states, or
3. Final $P(q,T')$ is incorrect [5]; i.e. the stable state reached after the occurrence of the fault does not satisfy the correctness conditions, irrespective of $P(q,T)$. In case of a fault-free sequence, where $T = T'$, the error is attributed to a protocol design error. Whereas when $T \neq T'$, and final $P(q,T)$ is correct, the error is manifested by the fault.

Since we are only concerned with the stable (i.e. non-transient) behavior of a protocol, we will only use the second and third definition for our study.

3.3 Test Input Pattern

A test input pattern is defined by a list of (host) events 'Ev', a topology 'T', and a fault model 'F', as shown in figure 2.

We define a test input pattern as a 3-tuple '$< Ev, T, F >$':

- Events: $Ev =< ev_1, ev_2, ..., ev_n >$ is a list of host events (host scenarios, or call patterns). Each event ev_j consists of $< action, time >$, where action: is the host (or node) event input; for example, join, leave, send packet, etc.

  Events maybe classified in many different ways, but just to show the extent of this dimension we only mention here triggered, timed and interleaved events.

[3] For PIM, being robust to a single message loss implies that transitions causing the protocol to move from one stable state to another be correct, even in the presence of single message loss. For the sake of analyzing erroneous behavior, however, we consider single message loss per test sequence. Test sequences and stable states are described in later sections.

[4] The fault may be empty in which case $T = T'$.

[5] Correctness is defined by the protocol specification. See section 3.4.

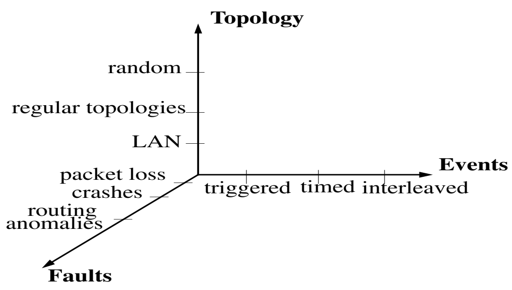

Figure 2: Test pattern dimensions

- Topology: $T =< N, L >$ is the routed topology of set of nodes N and links L. $N =< n_1, n_2, ...n_k >$, is the list of nodes each running a set of protocols. $L =< l_1, l_2, ...l_m >$ are the links connecting the nodes; two in case of a point-to-point link, or more for LANs.

  The topology dimension extends to cover LANs, regular topologies (e.g. star, string, ring, tree), and a mix thereof (i.e. random topologies).

- Faults: F is the fault model used to inject the fault into the test. According to the single-message loss model, for example, a fault may denote the 'loss of the second message traversing link l_i of type *prune*'. Knowing the location and the triggering action of the fault is important in analyzing the protocol behavior.

  Faults may include packet loss, crashes, and routing anomalies, among others.

3.4 PIM-DM

As a case study, we apply our automatic test generation method to a version of the Protocol Independent Multicast-Dense Mode (PIM-DM) [4] protocol.

PIM-DM uses broadcast-and-prune to establish the multicast distribution tree. In this mode of operation, a multicast packet is broadcast to all leaf subnetworks. Subnetworks with no local members send *prune* messages towards the source(s) of the packets to stop further broadcasts.

Routers with new members joining the group trigger *Graft* messages towards previously pruned sources to re-establish the branches of the delivery tree. *Graft* messages are acknowledged explicitly at each hop using the *Graft-Ack* message.

PIM-DM uses the underlying unicast routing tables to obtain the next-hop information needed for the RPF (reverse-path-forwarding) checks. This may lead to situations where there are multiple forwarders for a LAN. The *Assert* mechanism resolves these situations and ensures there is at most one forwarder for the LAN.

PIM Protocol Errors In this study we target protocol design and specification errors. We are interested mainly in erroneous stable (i.e. non-transient) states. We assume that these errors are provided by the protocol designer or specification. A protocol error may manifest itself in one of the following ways:

1. *black holes*: consecutive packet loss between periods of packet delivery.
2. *packet looping*: the same packet traverses the same set of links multiple times.
3. *packet duplication*: multiple copies of the same packet are received by the same receiver(s).
4. *join latency*: time taken by a receiver joining the group to start receiving packets destined to the group.
5. *leave latency*: time taken after a receiver leaves the group to stop the packets from flowing down the branches that no longer lead to receivers.

Some of these manifestations concern the correct delivery of packets, while others (e.g. leave latency) concern efficiency and conservation of network resources.

Correctness Conditions We assume that correctness conditions are provided by the protocol designer or specification. These conditions are necessary to avoid (during stable, non-transient states) the above protocol errors in a LAN environment, and are defined in terms of protocol states (as opposed to end-point behavior):

1. If one (or more) of the routers is expecting to receive packets from the link (i.e. having the link as their next-hop), then one other router must be a forwarder for the link. Violation of this condition may lead to data packet loss (e.g. join latency or black holes).
2. The link must have at most one forwarder at a time. Violation of this condition may lead to data packet duplication.
3. The delivery tree must be loop-free:
   (a) Any router should accept packets for (S,G) from one incoming interface only. This condition is enforced by the RPF (Reverse Path Forwarding) check.
   (b) The underlying unicast topology should be loop-free[6].

   Violation of this condition may lead to data packet looping.
4. If one of the routers is a forwarder for the link, then there must be at least one router expecting packets from the link (i.e. having the link as their next-hop). Violation of this condition may lead to leave latency.

3.5 The Protocol Model

As mentioned earlier, by virtue of being deterministic, fault-oriented test generation requires the definition of a protocol model. Formally, we present the protocol by a finite state machine (FSM), and the LAN by a global FSM model, as follows:

I. FSM model A deterministic finite state machine modeling the behavior of a router R_i is represented by the machine $\mathcal{M}_i = (\mathcal{Q}, \Sigma_i, \delta_i)$, where

- $\mathcal{Q}$ is a finite set of state symbols
- Σ_i is the set of operations causing state transitions; and
- δ_i is the state transition function $\mathcal{Q} \times \Sigma_i \rightarrow \mathcal{Q}$,

II. Global FSM model With respect to a particular LAN, the global state is defined as the composition of individual router states w.r.t. to that LAN. The behavior of a LAN with n routers may be described by the global FSM $\mathcal{M}_\mathcal{G} = (\mathcal{Q}_\mathcal{G}, \Sigma_\mathcal{G}, \delta_\mathcal{G})$ where

- $\mathcal{Q}_\mathcal{G}$: $\mathcal{Q}_1 \times \mathcal{Q}_2 \times \cdots \times \mathcal{Q}_n$ is the global state space,
- $\Sigma_\mathcal{G}$: $\bigcup_{i=1}^{n} \Sigma_i$ is the set of operations causing the transitions; and
- $\delta_\mathcal{G}$: is the global state transition function $\mathcal{Q}_\mathcal{G} \times \Sigma_\mathcal{G} \rightarrow \mathcal{Q}_\mathcal{G}$, which is defined as

$$\delta_\mathcal{G}\left((u_1, u_2, \ldots, u_n), (q_1, q_2, \ldots, q_n), x\right) \triangleq (\delta_1(u_1, q_1, x), \ldots, \delta_n(u_n, q_n, x))$$

[6]Some esoteric scenarios of route flapping may lead to multicast loops, in spite of RPF checks. Currently, our study does not address this issue, as it does not pertain to a localized behavior.

4 Applying the method

In this section, we investigate fault-oriented automatic test generation. Fault-oriented test generation (FOTG) targets specific faults. Starting from a given fault, FOTG attempts to synthesize minimal topology(ies) that may experience an error, and a sequence of events leading to the error.

The faults studied here are single message loss, and loss of state:

1. For single message loss the algorithm is run for a specific message, and is repeated for other protocol messages. For a given message, the algorithm identifies a set of stimuli and states needed to stimulate that message, and the possible states and stimuli elicited by the message. The set of states form the system state to be inspected, and the system components required to represent these states form a topology that may be vulnerable to error. The protocol model is used to derive this information.

   Similarly, for loss of state, the algorithm is repeated for each state. The topology necessary to create the state is constructed and the initial system state to be inspected is obtained.

2. Subsequent system states are obtained, through a process called *forward implication*, after the fault is included in the implication rules. Forward implication is the process of inferring subsequent states from a given state. The subsequent stable state is checked for errors.

3. If an error occurs, an attempt is made to obtain a sequence of events leading from an initial state to the inspected state, if such state is reachable. Such process is called *backward implication.*

Details of these algorithms are presented in section 4.5.

The rest of this section is organized as follows. The protocol model is presented and applied to PIM-DM in sections 4.1, 4.2, 4.3, 4.4, 4.5. Fault-oriented analysis of PIM-DM is given in section 4.6. Section 4.7 concludes our case study.

4.1 PIM-DM Model

We represent the protocol as a finite state machine (FSM), and extend it to capture the protocol LAN environment as a global FSM.

I. FSM model $\mathcal{M}_i = (\mathcal{Q}_i, \Sigma_i, \delta_i)$

Following is an FSM model of a simplified version of PIM-DM. For a specific source-group pair, we define the states w.r.t. a specific LAN to which the router R_i is attached. For example, a state may indicate that a router is a forwarder for (or receiver expecting packets from) the LAN.

A. System States ($\mathcal{Q}$) The possible states in which a router in the system may exist are described in the following table:

State Symbol	**Meaning**
F_i	Router i is a forwarder for the LAN
F_{i_Timer}	i forwarder with Timer $_{Timer}$ running
NF_i	Upstream router i is not a forwarder, but has entry
NH_i	Router i has the LAN as its next-hop
NH_{i_Timer}	same as NH_i with the Timer $_{Timer}$ running
NC_i	Router i has a negative-cache entry pointing to the LAN
EU_i	Upstream router i does not have an entry; i.e. is empty
ED_i	Downstream router i does not have an entry; i.e. is empty
M_i	Downstream leaf router with no state, and an attached member
NM_i	Downstream leaf router with no state and no attached members

The possible states for *upstream* and *downstream* routers are as follows:

$$\mathcal{Q}_i = \begin{cases} \{F_i, F_{i\_Timer}, NF_i, EU_i\} & \text{if the router is upstream,} \\ \{NH_i, NH_{i\_Timer}, NC_i, M_i, NM_i, ED_i\} & \text{if the router is downstream.} \end{cases}$$

B. Stimuli and Events (Σ) The stimuli and system events considered here include transmitting and receiving protocol messages, timer events, and external host events. Only events leading to change of state, or stimulation of other events are considered. For example, transmitting messages per se (vs. receiving messages) does not cause any change of state, unless it is an acknowledged message (e.g. a $Graft$), in which case the transmission causes the retransmission timer to be set. Following are the events considered in our study:

- Transmitting messages: Graft transmission ($Graft_{Tx}$)
- Receiving messages: Graft reception ($Graft_{Rcv}$), Join reception ($Join$), Prune reception ($Prune$), Graft Acknowledgement reception ($GAck$), Assert reception ($Assert$), and forwarded packets reception ($FPkt$).
- Timer events: these events occur due to timer expiration (Exp) and include the Graft re-transmission timer (Rtx), the event of its expiration ($RtxExp$), the forwarder-deletion timer[7] (Del), and the event of its expiration ($DelExp$).

  We note here that the expiration event of a timer is implied when a timer is set. This event is referred to as ($TimerImplication$).
- External host events: abbreviated as (Ext) include host sending packets ($SPkt$), host joining a group ($HJoin$ or HJ)), and host leaving a group ($Leave$ or L).

$$\Sigma = \{Join, Prune, Graft_{Tx}, Graft_{Rcv}, GAck, Assert, FPkt, Rtx, Del, SPkt, HJoin, Leave\}$$

II. Global FSM model An example global state for a topology of 4 routers connected to a LAN, with router 1 as a forwarder, router 2 expecting packets from the LAN, and routers 3 and 4 have negative caches, is given by $\{F_1, NH_2, NC_3, NC_4\}$.

4.2 Transition Table

The global state transition may be represented in several ways. Here, we choose a transition table representation that emphasizes the effect of the stimuli on the system, and hence facilitates topology synthesis, as will be shown. The transition table describes, for each stimulus or event, the conditions of its occurrence. A condition is given as stimulus, and state or transition, (denoted by *stimulus.state/trans*); where the transition is given as $startState \rightarrow endState$. A pre-condition for an event is a sufficient condition[8] to trigger the event. In contrast, a post-condition for a stimulus is an event and/or transition that is triggered by the stimulus, in the absence of faults (e.g. message loss). A '(p)' indicates a possible transition or stimulus, and represents a branching point in the search space. *orig*, *dst*, and *other* indicate the origin of the stimulus, its destination, and other routers, respectively. Following is the transition table for the global FSM discussed earlier.

Stimulus	**Pre-conditions** (stimulus.state/trans)	**Post-conditions** (simulus.state/trans)
$Join$	$Prune_{other}.NH_{orig}$	$F_{dst\_Del} \rightarrow F_{dst}, NF_{dst} \rightarrow F_{dst}$
$Prune$	$L.NC, FPkt.NC$	$F_{dst} \rightarrow F_{dst\_Del}$.(p)$Join_{other}$
$Graft_{Tx}$	$HJ.(NC \rightarrow NH), RtxExp.(NH\_{Rtx} \rightarrow NH)$	$Graft_{Rcv}.(NH \rightarrow NH\_{Rtx})$
$Graft_{Rcv}$	$Graft_{Tx}.(NH \rightarrow NH\_{Rtx})$	$GAck.(NF_{dst} \rightarrow F_{dst})$
$GAck$	$Graft_{Rcv}.F$	$NH_{dst\_Rtx} \rightarrow NH_{dst}$
$Assert$	$FPkt_{other}.F_{orig}, Assert_{other}.F_{orig}$	(p) $F_{other} \rightarrow NF_{other}$, (p)$Assert_{other}$
$FPkt$	$Spkt.F$	$Prune.(NM \rightarrow NC), ED \rightarrow NH, M \rightarrow NH,$ $EU_{other} \rightarrow F_{other}$,(p) $Assert$
Rtx	RtxExp	$Graft_{Tx}.(NH_{orig\_Rtx} \rightarrow NH_{orig})$
Del	DelExp	$F_{orig\_Del} \rightarrow NF_{orig}$
$SPkt$	Ext	$FPkt.(EU_{orig} \rightarrow F_{orig})$
$HJoin$	Ext	$NM \rightarrow M, Graft_{Tx}.(NC \rightarrow NH)$
$Leave$	Ext	$M \rightarrow NM, Prune.(NH \rightarrow NC),$ $Prune.(NH_{Rtx} \rightarrow NC)$

[7] This is referred to as Oif-Deletion timer in the PIM specification.

[8] Here, we use the term *sufficient* loosely. It is actually also necessary to have one pre-condition (say X) to trigger the stimulus (Y) such that $X \Leftrightarrow Y$ holds and we can construct the backward implication rules. If more than a pre-condition (say X_1, X_2) is specified, we may infer either $Y \Rightarrow X_1$ or $Y \Rightarrow X_2$, and this represents a branching point in the backward search.

4.3 State Dependency Table

To aid in test sequence synthesis through the backward implication procedure, we construct what we call a state dependency table. This table does not contain additional information about the protocol behavior to that given by the transition table, and is inferred automatically therefrom. We use this table to improve the performance of the algorithm and for illustration.

For each state, the dependency table contains the possible preceding states and the stimulus from which the state can be reached or implied. To obtain this information for a state S, we search the post-condition column of the transition table for entries where the *endState* of a transition is S. In addition, a state may be identified as an initial state (I.S.). The initial states for this study include $\{EU, ED, NM\}$.

Based on the above transition table, following is the resulting state dependency table:

State	Possible Backward Implication(s)
F_i	$\xleftarrow{Fpkt_{other}} EU_i, \xleftarrow{Join} F_{i\_Del}, \xleftarrow{Join} NF_i, \xleftarrow{Graft_{Rcv}} NF_i, \xleftarrow{SPkt} EU_i$
F_{i_Del}	$\xleftarrow{Prune} F_i$
NF_i	$\xleftarrow{Del} F_{i\_Del}, \xleftarrow{Assert} F_i$
NH_i	$\xleftarrow{Rtx,GAck} NH_{i\_Rtx}, \xleftarrow{HJ} NC_i, \xleftarrow{FPkt} M_i, \xleftarrow{FPkt} ED_i$
NH_{i_Rtx}	$\xleftarrow{Graft_{Tx}} NH_i$
NC_i	$\xleftarrow{FPkt} NM_i, \xleftarrow{L} NH_{i\_Rtx}, \xleftarrow{L} NH_i$
EU_i	$\leftarrow I.S.$
ED_i	$\leftarrow I.S.$
M_i	$\xleftarrow{HJ} NM_i$
NM_i	$\xleftarrow{L} M_i, \leftarrow I.S.$

4.4 Defining stable states

As mentioned earlier, we are concerned with stable state (i.e. non-transient) behavior. To establish the erroneous stable states, we need to define the transition mechanisms between such states, and so we introduce the concept of transition classification and completion to distinguish between transient and stable states.

Classification of Transitions We identify two types of transitions; *externally triggered (ET)* and *internally triggered (IT)* transitions. The first type is stimulated by actions external to the system (e.g. $HJoin$ or $Leave$), whereas the second type is stimulated by actions internal to the system (e.g. $FPkt$ or $Graft$).

We note that some transitions may be triggered due to both internal and external actions, depending on the scenario. For example, a $Prune$ may be triggered due to forwarding packets by an upstream router $FPkt$ (which is an internal action), or a $Leave$ (which is an external event).

The global state is checked for correctness only at the end of an externally triggered transition and after completing all its dependent internally triggered transitions.

Following is a table of host events, their dependent ET events and their dependent IT events:

Host Events	$SPkt$	$HJoin$	$Leave$
ET events	$FPkt$	$Graft$	$Prune$
IT events	$Assert$, $Prune$, $Join$	$GAck$	$Join$

Transition Completion To check for the global system correctness, all stimulated internal transitions should be completed, to bring the system into a stable state. Intermediate (transient) states should not be checked for correctness (since they may violate the correctness conditions set forth for stable states, and hence may give false error indication).

The process of identifying complete transitions depends on the nature of the protocol. But, in general, we may identify a complete transition sequence, as the sequence of (all) transitions triggered due to a single external stimulus (e.g. $HJoin$ or $Leave$). Therefore, we should be able to identify a transition based upon its stimuli (either external or internal).

At the end of each complete transition sequence the system exists in either a correct or erroneous stable state. Action-triggered timers (e.g. Del, Rtx) fire at the end of a complete transition, satisfying the $TimerImplication$.

Also, according to the above completion concept, the proper analysis of behavior should start from externally triggered transitions. For example, analysis should not consider a *Join* without considering the *Prune* triggering it and its effects on the system. Thus the global system state must be rolled back to the beginning of a complete transition (i.e. the previous stable state) before applying the forward implication. This will be implied in the forward implication algorithm discussed later, to simplify the discussion.

4.5 FOTG details

As previously mentioned, our FOTG approach consists of three phases: I) synthesis of the global state to inspect, II) forward implication, and III) backward implication. These phases are explained in more detail in this section.

FOTG starts from a given fault. The faults we address here are message and state loss.

Synthesizing the Global State

Starting from a message (i.e. the message or state to be lost), and using the information in the protocol model (i.e. the transition table), a global state is chosen for investigation. We refer to this state as the global-state inspected (G_I), and it is obtained for message loss as follows:

1. The global state is initially empty and the inspected message is initially set to the message to be lost.
2. For the inspected message, the state (or the *startState* of the transition) of the post-condition is obtained from the transition table. If the state does not exist in the global state, and cannot be implied therefrom, then it is added to the global state.
3. For the inspected message, the state (or the *endState* of the transition) of the pre-condition is obtained. If the state does not exist in the global state, and cannot be implied therefrom, then it is added to the global state.
4. Get the stimulus of the pre-condition of the inspected message. If this stimulus is not external (*Ext*), then set the inspected message to the stimulus, and go back to step 2.

At the end of this stage, the global state to be investigated is obtained[9].

For state loss, the state dependency table is used to determine the message required to create the state, and the topology constructed for that message is used for the state. This is illustrated in section 4.6.3.

Forward Implication

The states following G_I (i.e. G_{I+i} where $i > 0$) are obtained through forward implication. We simply apply the transitions, starting from G_I, as given by the transition table, in addition to implied transitions (such as timer implication). In case of a message loss, the transition due to the lost message is not applied. If more than a state is affected by the message, then the space searched is expanded to include the various selective loss scenarios for the affected routers.

Backward Implication

If an error occurs, backward implication attempts to obtain a sequence of events leading to G_I, from an initial state ($I.S.$), if such sequence exists; i.e. if G_I is reachable from $I.S.$

The state dependency table is used in the backward search. For each component in the global state G_I, backward steps are taken[10] until an initial global state (i.e. a state with all components as $I.S.$) is reached[11].

Figure 3 shows the above processes for a simple example.

4.6 Fault-oriented Analysis for PIM-DM

In this section we discuss the results of applying our method to PIM-DM. The analysis is conducted for single message loss and momentary loss of state.

[9] Note that although in lots of cases the topology will be constructed during the first phase of obtaining G_I, the topology may still be expanded during the forward/backward implication phases.

[10] For each backward step, a forward step is taken to verify that interaction between components does not lead to a state different than that from which the backward step was taken. This may be optimized if we are certain that such interaction will not occur. We do not, however, discuss such optimization here, we consider it part of our future work.

[11] A termination condition (such as max. number of states investigated) may be used to terminate the algorithm. The details of such condition are irrelevant in this context.

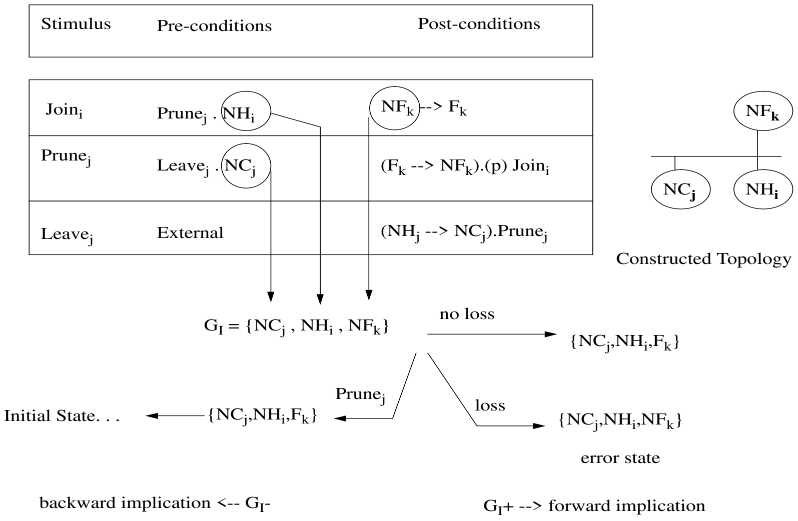

Figure 3: Simple example for topology synthesis, forward implication and backward implication for *Join*

4.6.1 Single message loss

We have studied single message loss scenarios for the $Join, Prune, Assert$, and $Graft$ messages. For brevity, we partially discuss our results here. For this subsection, we consider non-interleaving external events, where the system is stimulated only once between stable states. The $Graft$ message is particularly interesting, since it is acknowledged, and it raises timing and sequencing issues that we address in a later subsection, where we extend our method to consider interleaving of external events.

For the following analyzed messages, we present the steps for topology synthesis, forward and backward implication.

Join: Following are the resulting steps for *join* loss:

Join Loss	Synthesizing the Global State
	1. set the inspected message to $Join$
	2. the *startState* of the post-condition is $F_{dst\_Del} \Longrightarrow G_I = \{F_{j\_Del}\}$
	3. the state of the pre-condition is $NH_i \Longrightarrow G_I = \{NH_i, F_{j\_Del}\}$
	4. the stimulus of the pre-condition is $Prune$. Set the inspected message to $Prune$
	5. the *startState* of the post-condition is F_j which can be implied from F_{j_Del} in G_I
	6. the state of the pre-condition is $NC_k \Longrightarrow G_I = \{NH_i, F_{j\_Del}, NC_k\}$
	7. the stimulus of the pre-condition is L. Set the inspected message to L
	8. the *startState* of the post-condition is NH which can be implied from NC in G_I
	9. the state of the pre-condition is Ext, an external event
	Forward implication
	without loss: $G_I = \{NH_i, F_{j\_Del}, NC_k\} \xrightarrow{Join} G_{I+1} = \{NH_i, F_j, NC_k\}$ correct state
	loss w.r.t. affected routers (i.e. R_j): $\{NH_i, F_{j\_Del}, NC_k\} \xrightarrow{Del} G_{I+1} = \{NH_i, NF_j, NC_k\}$ error state
	Backward implication
	$G_I = \{NH_i, F_{j\_Del}, NC_k\} \xleftarrow{Prune} G_{I-1} = \{NH_i, F_j, NC_k\} \xleftarrow{FPkt} G_{I-2} = \{M_i, F_j, NM_k\}$ $\xleftarrow{SPkt} G_{I-3} = \{M_i, EU_j, NM_k\} \xleftarrow{HJ_i} G_{I-4} = \{NM_i, EU_j, NM_k\} = I.S.$

Losing the $Join$ by the forwarding router R_j leads to an error state where router R_i is expecting packets from the LAN, but the LAN has no forwarder.

Assert: Following are the resulting steps for the $Assert$ loss:

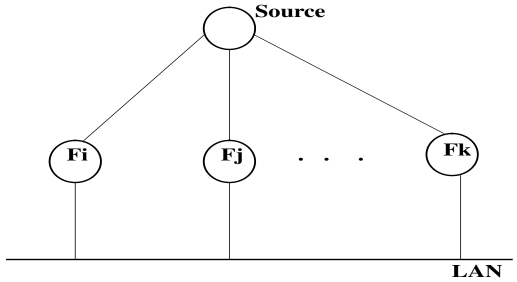

Figure 4: A topology having a $\{F_i, F_j, \ldots, F_k\}$ LAN

Assert Loss	**Synthesizing the Global State**
	1. set the inspected message to $Assert$
	2. the *startState* of the post-condition is $F_j \Longrightarrow G_I = \{F_j\}$
	3. the state of the pre-condition is $F_i \Longrightarrow G_I = \{F_i, F_j\}$
	4. the stimulus of the pre-condition is $FPkt_j$. Set the inspected message to $FPkt_j$
	5. the *startState* of the post-condition is EU_i, which can be implied from F_i in G_i
	6. the state of the pre-condition is F_j, already in G_I
	7. the stimulus of the pre-condition is $SPkt_j$. Set the inspected message to $SPkt_j$
	8. the *startState* of the post-condition is NF_j, which can be implied from F_j in G_I
	9. the stimulus of the pre-condition is Ext, an external event
	Forward Implication
	$G_I = \{F_i, F_j\} \xrightarrow{Assert_i} G_{I+1} = \{F_i, NF_j\}$ error
	Backward Implication
	$G_I = \{F_i, F_j\} \xleftarrow{FPkt_j} G_{I-1} = \{EU_i, F_j\} \xleftarrow{Spkt_j} G_{I-2} = \{EU_i, EU_j\} = I.S.$

The error in the $Assert$ case occurs even in the absence of message loss. This error occurs due to the absence of a prune to stop the flow of packets to a LAN with no downstream receivers. This problem occurs for topologies with $G_I = \{F_i, F_j, \ldots, F_k\}$, as that shown in figure 4.

Graft: Following are the resulting steps for the $Graft$ loss:

Graft Loss	**Synthesizing the Global State**
	1. Set the inspected message to $Graft_{Rcv}$
	2. the *startState* of the post-condition is $NF \Longrightarrow G_I = \{NF\}$
	3. the *endState* of the pre-condition is $NH_{Rtx} \Longrightarrow G_I = \{NF, NH_{Rtx}\}$
	4. the stimulus of the pre-condition is $Graft_{Tx}$
	5. the *startState* of the post-condition is NH, which may be implied from NH_{Rtx} in G_I
	6. the *endState* of the pre-condition is NH which may be implied
	7. the stimulus of the pre-condition is HJ, which is Ext; i.e. external
	Forward Implication
	without loss: $G_I = \{NH, NF\} \xrightarrow{Graft_{Tx}} G_{I+1} = \{NH_{Rtx}, NF\} \xrightarrow{Graft_{Rcv}} G_{I+2} = \{NH_{Rtx}, F\} \xrightarrow{GAck} G_{I+3} = \{NH, F\}$ correct state
	with loss of $Graft$; i.e. the $Graft_{Rcv}$ does not take effect: $G_I = \{NH, NF\} \xrightarrow{Graft_{Tx}} G_{I+1} = \{NH_{Rtx}, NF\} \xrightarrow{TimerImplication} G_{I+2} = \{NH, NF\} \xrightarrow{Graft_{Tx}} G_{I+3} = \{NH_{Rtx}, NF\} \xrightarrow{Graft_{Rcv}} G_{I+4} = \{NH_{Rtx}, F\} \xrightarrow{GAck} G_{I+5} = \{NH, F\}$ correct state

We did not reach an error state when the $Graft$ was lost, with non-interleaving external events.

4.6.2 Interleaving events and Sequencing

A $Graft$ message is acknowledged by the $Graft - Ack$ ($GAck$) message, and is inherently robust to message loss according to the completion and timer implication conditions, given no other external adverse events interrupt these

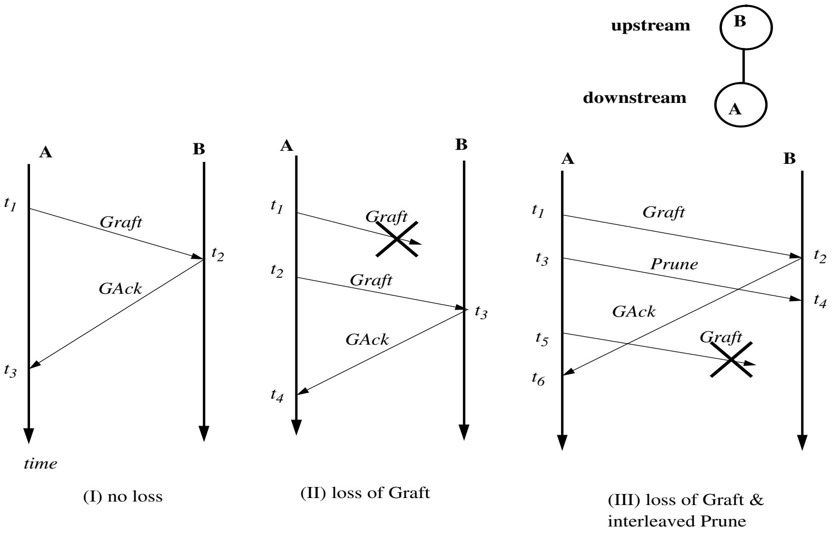

Figure 5: Graft event sequencing

conditions.

To examine the vulnerability of acknowledged messages, we try to interleave adversary external conditions during the transient states in which the system exists and before the completion of the acknowledged message phase. To achieve this, we clear the retransmission timer, such that the adverse event will not be overridden by the retransmission mechanism.

To clear the retransmission timer, we should create a transition from NH_{Rtx} to NH which is triggered by a $GAck$ according to the state dependency table ($NH \xleftarrow{GAck} NH_{Rtx}$). We then insert this transition in the event sequence.

Forward Implication $G_I = \{NH, NF\} \xrightarrow{Graft_{Tx}} G_{I+1} = \{NH_{Rtx}, NF\} \xrightarrow{GAck} G_{I+2} = \{NH, NF\}$ error state.

Backward Implication Using backward implication, we can construct a sequence of events leading to conditions sufficient to trigger the $GAck$. From the transition table these conditions are $\{NH_{Rtx}, F\}$[12]:
$G_I = \{NH, NF\} \xleftarrow{HJ} G_{I-1} = \{NC, NF\} \xleftarrow{Del} G_{I-2} = \{NC, F_{Del}\} \xleftarrow{Prune} G_{I-3} = \{NC, F\} \xleftarrow{L} G_{I-4} = \{NH_{Rtx}, F\}$.

To generate the $GAck$ we continue the backward implication and attempt to reach an initial state:
$G_{I-4} = \{NH_{Rtx}, F\} \xleftarrow{GraftRcv} G_{I-5} = \{NH_{Rtx}, NF\} \xleftarrow{Graft_{Tx}} G_{I-6} = \{NH, NF\} \xleftarrow{HJ} G_{I-7} = \{NC, NF\} \xleftarrow{Del} G_{I-8} = \{NC, F_{Del}\} \xleftarrow{Prune} G_{I-9} = \{NC, F\} \xleftarrow{FPkt} G_{I-10} = \{NM, F\} \xleftarrow{SPkt} G_{I-11} = \{NM, EU\} = I.S.$

The overall sequence of events is illustrated in figure 5. In the first and second scenarios (**I** and **II**) no error occurs, as the retransmission timer takes care of the single $Graft$ loss. However, in the third scenario (**III**) when a $Graft$ followed by a $Prune$ is interleaved with the $Graft$ loss, the retransmission timer is reset with the receipt of the $GAck$ for the first $Graft$, and the systems ends up in an error state.

4.6.3 Loss of State

We consider momentary loss of state to a router on the LAN. A '$Crash$' stimulus transfers any state 'X' into 'EU' ($X \xrightarrow{Crash} EU$ for upstream router) or 'ED' ($X \xrightarrow{Crash} ED$ for downstream router). Hence, we add the following line to the transition table:

Stimulus	**Pre-conditions** (stimulus.state/trans)	**Post-conditions** (simulus.state/trans)
$Crash$	Ext	$NM, M, NH, NC, NH_{Rtx} \rightarrow ED$, $F, F_{Del}, NF \rightarrow EU$

[12]We do not show all branching or backtracking steps for simplicity.

Since a $Crash$ can occur at any point in time, the transition table must be completed to specify the action taken (if any) for an empty state upon receiving any message. For momentary loss of state, the FSM resumes function immediately after the transition to the empty state (i.e. further transitions are not affected by the crash).

To study the effect of this type of crash on the protocol, we analyze the behavior when the crash occurs in any router state. For every state, a topology is synthesized that is necessary to create that state. We leverage the topologies previously synthesized for the messages. For example, state F_{Del} may be created from state F by receiving a $Prune$ (check the dependency table entry for $F_{Del} \xleftarrow{Prune} F$). Hence we may use the topologies constructed for $Prune$ loss to analyze a crash for F_{Del} state.

Forward implication is then applied (along with the crash). The behavior of the system after the crash is inspected to see if packet delivery is affected. To achieve this, host stimuli (i.e. $SPkt$, HJ and L) are applied to the routers, then the system state is checked for correctness.

In lots of the cases studied, the system recovered from the crash (i.e. the system state was eventually correct). An example of a recovery process is given in figure 6. The recovery is mainly due to the nature of PIM-DM; where protocol states may be created from the empty state with the reception of data packets. This result is not likely to extend to other multicast routing protocols of other natures (such as PIM Sparse-Mode [33]).

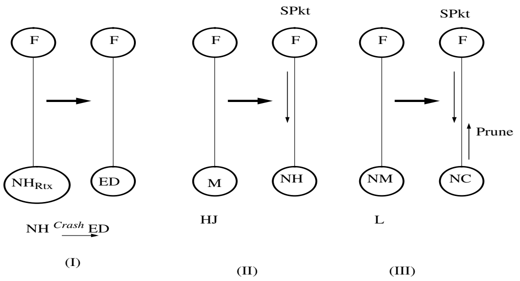

Figure 6: System recovering from a crash

However, in other cases the system did not recover, we discuss some of these cases briefly here. The first scenario is given in figure 7, where the host joining in (II, a) does not have the sufficient state to send a $Graft$ and hence gets join latency until the negative cache state times out upstream and packets are forwarded onto the LAN as in (II, b).

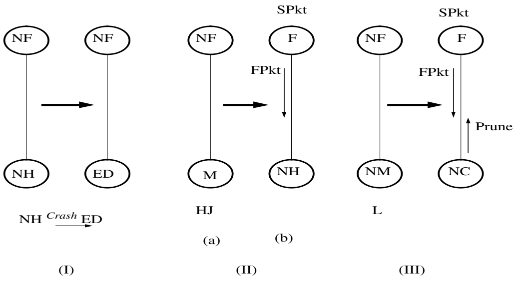

Figure 7: Crash leading to join latency

The scenario in figure 8 (II, a) shows the downstream router black-holed due to the crash of the upstream router. The state is not corrected until the periodic broadcast takes place, and packets are forwarded onto the LAN as in (II, b).

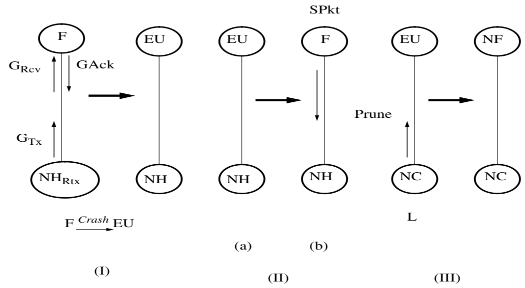

Figure 8: Crash leading to black holes

4.7 Case Study Conclusion

Even with the simple study presented above, we were able to drive a protocol (which has been deployed in parts of the Internet for over two years) into error states, by considering scenarios of single packet loss or crashes. We were also able to construct erroneous scenarios for acknowledged messages (such as $Graft$) and analyze the cause of the error. Automation was needed throughout this process, to tackle the complexity of protocol behavior and robustness analysis.

We believe that the strength of our fault-oriented method, as was demonstrated, lies in its ability to construct the necessary conditions for erroneous behavior by starting directly from the fault, and avoiding the exhaustive walk of the state space. Also, converting timing problems into sequencing problems (as was shown for $Graft$ analysis) reduces the complexity required to study timers.

Since network dynamics and failures strongly affect protocol behavior, as we have shown, we think that robustness studies should be integrated with the protocol design, and not just considered as an after thought.

Although we have not presented a complete list of our findings, we are encouraged by the cases presented in this paper to pursue our research in applying our method to other multicast protocols and applications.

5 Summary and Future Work

We have presented a new method for automating the testing and analysis of multicast routing protocol robustness. Our approach attempts to provide a practical tool for studying real Internet multicast protocols in the presence of network failures and packet loss.

We do not claim nor attempt to provide mathematical proof of protocol correctness or verification. Rather, we have targeted protocol robustness and endeavored to systematize its testing and analysis for a particular domain; multicast routing.

Drawing from well-established chip testing techniques and FSM modeling, our method synthesizes the protocol tests automatically. These tests consist of the topology, event sequences and network faults. The following techniques were used to automate each of the test dimensions:

- Topology synthesis: we have used a model for LAN topology with N routers (where N is variable). Using the state transition table, our method synthesizes topologies necessary to generate protocol messages or states. The global state of the system to be inspected is obtained, to be analyzed in later stages.
- Fault investigation: starting from the state of the system to be inspected, a forward implication technique is used to test the behavior of the system in the presence of faults. Faults investigated in this study include selective single message loss, and momentary loss of state.
- Sequence of events: if an error is found, a backward implication technique establishes a sequence of events leading to the erroneous state (if it is reachable) from an initial state. This sequence is used to analyze the protocol behavior leading to the error, and may be used to drive more detailed simulations and tests.
- Timing problems: during the process of message loss analysis, we have presented an example of transforming a timing problem into a sequencing problem to analyze acknowledged messages.

The analysis was presented in the context of a case study for PIM-DM. We found several scenarios of single packet loss and crashes in which the protocol behaved erroneously. We were able to construct a scenario of interleaving events that lead an acknowledged message mechanism into error in the presence of single message loss.

Our method may also be applicable to other protocols that can be represented by the global FSM model given in this paper.

Future directions of this research include[13] applying the FOTG method to other multicast routing protocols, such as PIM-SM and BGMP, to analyze their robustness. We will also investigate extending the method to apply to topologies containing multiple LANs, and to include other network failures such as unicast routing oscillations and flapping.

Another rich area is to use the FOTG method to analyze the performance of end-to-end multicast protocols (such as multicast transport protocols, and multicast applications) in a more systematic fashion. A logical network model may be used, where the multicast distribution tree is modeled by delay and loss matrices. This model may then be integrated with the global FSM model presented in this paper, and analyzed using a discrete event simulator.

[13] Mathematical treatment of the formalism, completeness, or complexity is out-of-scope of this paper, and belongs in other work we plan to publish. This paper presents the basic algorithmic principles, and demonstrates the utility of the method in real Internet protocol design.

References

[1] V. Paxon. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking, Vol. 5, No. 5. An earlier version appeared in Proc. ACM SIGCOMM '96, Stanford, CA*, pages 601–615, October 1997.

[2] V. Paxon. End-to-End Internet Packet Dynamics. *ACM SIGCOMM '97*, September 1997.

[3] A. Helmy and D. Estrin. Simulation-based STRESS Testing Case Study: A Multicast Routing Protocol. *Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)*, July 1998.

[4] 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.

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

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

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

[8] 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.

[9] 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.

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

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

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

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.*

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

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

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

[22] 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.

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

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] F. Pong and M. Dubois. Verification Techniques for Cache Coherence Protocols. *ACM Computing Surveys, Volume 29, No. 1*, pages 82–126, March 1996.

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

[33] 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.