FAULT-ORIENTED TEST GENERATION FOR MULTICAST ROUTING PROTOCOL DESIGN

Ahmed Helmy, Deborah Estrin, Sandeep Gupta

University of Southern California
Los Angeles, CA 90089
{ahelmy,estrin}@usc.edu, sandeep@boole.usc.edu

Abstract: We present a new algorithm for automatic test generation for multicast routing. 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 target protocol robustness in specific, and do not attempt to verify other properties in this paper. We apply our method to a multicast routing protocol; PIM-DM, and investigate its behavior in the presence of selective packet loss on LANs and router crashes. Our study unveils several robustness violations in PIM-DM, for which we suggest fixes with the aid of the presented algorithm.

1.1 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, such as IP multicast. 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 network components.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35394-4_29

S. Budkowski et al. (eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*

Many researchers have developed protocol verification methods, to ensure that certain properties of a protocol hold. Much of this work, however, was based on abstract mathematical models, with assumptions about the network conditions that may not always hold in today's Internet. To date, little effort has been exerted to formulate practical methods and tools that aid in the systematic testing of multicast protocols.

In this study, we propose a new method for automatic test generation, geared towards the study of protocol robustness in the presence of packet loss and network failures. In particular, we try to answer the question "is the protocol robust to specific failures ?" However, we do not attempt to develop a general verification method.

We borrow from well-established chip testing technologies and apply them to network protocols. We refer to our method as the fault-oriented test generation (FOTG). Starting from a given fault, the necessary topology and event sequences are established, that drive the protocol into error states, using forward and backward search techniques. As a case study, we apply FOTG to a real multicast routing protocol; PIM-DM [4], that has been deployed in parts of the Internet. 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 [3].

The rest of this paper is organized as follows. Related work is discussed in section 1.2. Section 1.3 provides method overview and definitions. Section 1.4 describes our algorithm in detail, and summarizes the results of our case study. Conclusion is given in section 1.5.

1.2 RELATED WORK

There is a large body of literature dealing with verification of protocols. Verification systems typically address well-defined properties –such as *safety* (e.g. deadlock freedom), *liveness* (e.g. livelock freedom), and *responsiveness* (e.g. timeliness) [5]– and aim to detect violations of these properties.

In general, the two main approaches for protocol verification are theorem proving and reachability analysis [6]. Theorem proving systems define a set of axioms and relations to prove properties mathematically. Theorem proving includes *model-based* (e.g. Z [7]) and *logic-based* formalisms (e.g. Nqthm [8]). In general, however, the number of axioms and relations 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 specifications, and hence tend to abstract out some network dynamics that we will study; such as selective packet loss and router crashes.

Reachability analysis algorithms [9], on the other hand, try to generate and inspect all reachable protocol states. Such algorithms suffer from the 'state space explosion' problem, especially for complex protocols. To circumvent this problem, state reduction and partial search techniques [10] could be used. These algorithms, however, do not synthesize network topologies. Reduced reachability analysis has been used in the verification of cache coherence protocols [11],

using a global FSM model. We adopt a similar FSM model and extend it for our approach in this study.

In [3] we have proposed a simulation-based STRESS testing method, based on heuristics and topological equivalences to reduce the number of simulated scenarios. However, we did not address automatic generation of topologies and events. Work in this paper complements our previous work, and may be integrated with the STRESS framework as part of its scenario generation.

VLSI chip testing [12] uses test vector generation to detect single-stuck faults. Test vectors may be generated based on circuit and fault models, using the fault-oriented process, that utilizes *implication* techniques for line justification. 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 a new dimension to the test generation problem.

1.3 METHOD OVERVIEW AND DEFINITIONS

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; e.g. packet loss, or machine crashes. The desired output is a set of test-suites that stress the protocol mechanisms according to the robustness criteria.

Our method produces tests based on a model of the protocol. This section describes the model used and gives an overview of the case study protocol; PIM-DM.

1.3.1 System Model and Test Definition

The system consists of network and topology elements and a fault model.

Elements of the network consist of multicast capable nodes and bi-directional symmetric links. Nodes run same multicast routing, but not necessarily the same unicast routing. The topology is a N-router LAN modeled at the network level; we do not model the MAC layer.

A *fault*, is a low level (e.g. physical layer) anomalous behavior, that may affect the protocol under test. An *error* is the failure of a protocol to meet its design requirement (e.g. duplicate packet delivery).

Faults include: a) Loss of packets due to queue congestion or failure. b) Loss of state, e.g. uni/multicast tables, due to failures or crashes. Loss duration varies with the nature of the failure. c) Delays due to transmission, propagation, or queuing delays. Some delay problems may be translated into sequencing problems (see section 1.4.6).

Usually, a fault model is defined in conjunction with the protocol's robustness criteria. A design requirement for PIM is to be robust to single protocol message loss, which implies correct transitions from one stable state to another, even

in the presence of single message loss. We also study the momentary loss of state. To analyze erroneous behavior, we consider single message loss per test sequence.

A test input pattern is defined as a 3-tuple '$< Topology, Events, Faults >$'; where Events is a sequence of host events; e.g. join, leave, or send packet, and the topology and faults as defined above.

Test Sequence Definition. Given sequences $T =< e_1, e_2, \ldots, e_n >$ and $T' =< e_1, e_2, \ldots, e_j, f, e_k, \ldots, e_n >$, where e_i is an event and f is a fault. Let $P(q,T)$ be the sequence of states and stimuli of protocol P under test T starting from the initial state q. T' is a test sequence if final $P(q,T')$ is incorrect; i.e. the stable state reached after the occurrence of the fault does not satisfy the protocol correctness conditions (see section 1.3.2) 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. This definition ignores transient protocol behavior.

1.3.2 PIM-DM

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

Multicast routing protocols deliver packets efficiently to group members by establishing distribution trees. PIM-DM uses broadcast-and-prune to establish the 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, which 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.

In this study we target protocol robustness errors. We are interested mainly in erroneous stable (i.e. non-transient) states. We assume that protocol errors and correctness conditions are provided by the specification.

PIM Protocol Errors: 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*. 3) *packet duplication*. 4) *join latency*: excessive time taken by a receiver to start receiving packets. 5) *leave latency*: excessive time taken after a receiver leaves the group to stop the packet flow down pruned branches.

Correctness Conditions: These conditions are necessary to avoid errors during stable states in a LAN environment: 1) If there exists routers expecting packets from the LAN then there must exist a forwarder for the LAN, to pre-

vent data loss (e.g. join latency or black holes). 2) The LAN must have at most one forwarder at a time, to prevent duplicates. 3) If there exists no routers expecting packets from the LAN there must not exist a forwarder for the LAN, to prevent leave latency. 4) The delivery tree must be loop-free. We do not consider looping in this study, as it is not a local behavior.

1.3.3 The Protocol Model

We represent the protocol by a finite state machine (FSM), and the LAN by a global FSM model (GFSM), 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 stimuli 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. 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 stimuli causing the transitions; and $\delta_\mathcal{G}$: is the global state transition function $\mathcal{Q}_\mathcal{G} \times \Sigma_\mathcal{G} \rightarrow \mathcal{Q}_\mathcal{G}$,

1.4 APPLYING THE METHOD

Fault-oriented test generation (FOTG) targets specific faults. Starting from a given fault, FOTG attempts to synthesize 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 a given message, the algorithm uses the protocol model to identify a set of stimuli and states needed to stimulate that message, and the possible states and stimuli elicited by the message. This set of states form the global system state to be inspected. The algorithm is repeated for each message.

For loss of state, the global state is constructed from the mechanisms necessary to create the state, and the algorithm is repeated for each state.

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 error state, if such state is reachable. Such process is called *backward implication.*

Details of these algorithms are presented in section 1.4.5.

1.4.1 PIM-DM Model

Following is the model of a simplified version of PIM-DM.

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

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 ($\mathcal{Q}$). Possible states in which a router may exist are:

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 (Σ). The stimuli considered here include transmitting and receiving protocol messages, timer events, and external host events. Only stimuli leading to change of state are considered. For example, transmitting messages per se (vs. receiving messages) does not cause any change of state, except for the *Graft*, in which case the *Rtx* timer is set. Following are the stimuli considered in our study:

1. Transmitting messages: Graft transmission ($Graft_{Tx}$)

2. Receiving messages: Graft reception ($Graft_{Rcv}$), Join reception (*Join*), Prune reception (*Prune*), Graft Acknowledgement reception (*GAck*), Assert reception (*Assert*), and forwarded packets reception (*FPkt*).

3. 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 (*Del*), and the event of its expiration (*DelExp*). The expiration of a timer is implied as (*TimerImplication*) when the timer is set.

4. External host events (*Ext*): include host sending packets (*SPkt*), host joining a group (*HJoin* or *HJ*), and host leaving a group (*Leave* or *L*).

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

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 $\{F_1, NH_2, NC_3, NC_4\}$.

1.4.2 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. The transition table describes, for each stimulus, 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 $startState \rightarrow endState$. At least one pre-condition is necessary to trigger the stimulus. 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. For example, a *Join* reception: a) is caused by the reception of a *Prune* from another router, with the originator of the *Join* in NH state, and b) causes *dst* to transit into F_{dst} if it exists in F_{Del} or NF state (other transitions are left out for simplicity). Following is the transition table for the global FSM discussed earlier.

Stimulus	**Pre-cond** (stimulus.state/trans)	**Post-cond** (stimulus.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	$M \rightarrow NM, Prune.(NH \rightarrow NC)$, $Prune.(NH_{Rtx} \rightarrow NC)$

1.4.3 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 $\{EU, ED, NM\}$.

Following is a partial 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	$\xleftarrow{FPkt} NM_i, \xleftarrow{L} NH_{i\_Rtx}, \xleftarrow{L} NH_i$
$\vdots$	

In some cases, multiple states need to be simultaneously implied in one backward step, otherwise an $I.S.$ may not be reached. To do this, the transitions in the post-conditions of the stimulus are traversed, and any states in the global state that are *endState*s are replaced by their corresponding *startState*s. For example, $\{M_i, NM_j, F_k\} \xleftarrow{FPkt} \{NH_i, NC_j, F_k\}$.

1.4.4 Defining stable states

As mentioned earlier, we are concerned with stable state (i.e. non-transient) behavior. To obtain erroneous stable states, we need to define the transition mechanisms between such states. 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 former is stimulated by events external to the system (e.g. *HJoin* or *Leave*), whereas the latter is stimulated by events internal to the system (e.g. $FPkt$ or $Graft$).

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

A global state is checked for correctness at the end of an externally triggered transition after completing 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. Event-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.

1.4.5 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 fault (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.

Note that there may be several pre-conditions or post-conditions for a stimulus, in which case several choices can be made. These represent branching points in the search space.

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

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 later in this section.

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 one 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. Backward steps are taken for the components in the global state G_I, until an initial global state (i.e. a state with all components as $I.S.$) is reached, or the termination criteria (if any) is met.

Figure 1.1 shows the above processes for a simple example of a *Join* loss. Following are the steps taken for that example:

Synthesizing the Global State
1. *Join*: *startState* of the post-condition is $NF_{dst} \Longrightarrow G_I = \{NF_k\}$
2. *Join*: state of the pre-condition is $NH_i \Longrightarrow G_I = \{NH_i, NF_k\}$, goto *Prune*
3. *Prune*: *startState* of the post-condition is F_k which can be implied from NF_k in G_I
4. *Prune*: state of the pre-condition is $NC_j \Longrightarrow G_I = \{NH_i, NF_k, NC_j\}$, goto L
5. the *startState* of the post-condition is NH which can be implied from NC in G_I

Forward implication
without loss: $G_I = \{NH_i, NF_k, NC_j\} \xrightarrow{Join} G_{I+1} = \{NH_i, F_k, NC_j\}$
loss w.r.t. R_j: $\{NH_i, NF_k, NC_j\} \longrightarrow G_{I+1} = \{NH_i, NF_k, NC_j\}$ error
Backward implication
$G_I = \{NH_i, NF_k, NC_j\} \xleftarrow{Prune} G_{I-1} = \{NH_i, F_k, NC_j\} \xleftarrow{FPkt} G_{I-2} = \{M_i, F_k, NM_j\} \xleftarrow{SPkt} G_{I-3} = \{M_i, EU_k, NM_j\} \xleftarrow{HJ_i} G_{I-4} = \{NM_i, EU_k, NM_j\} = I.S.$

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

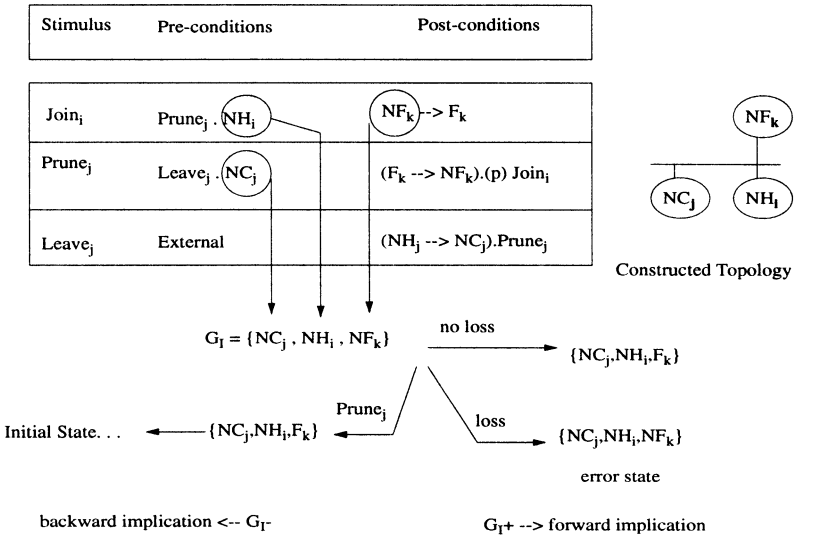

Figure 1.1 *Join* topology synthesis, forward/backward implication

1.4.6 Summary of Results

We have implemented an early version of the algorithm in the NS/VINT environment (see http://catarina.usc.edu/vint) and used it to drive detailed simulations of PIM-DM therein, to verify our findings. In this section we briefly discuss the results of applying our method to PIM-DM. The analysis is conducted for single message loss and momentary loss of state. For a detailed analysis of the results see [13].

Single message loss. We have studied single message loss scenarios for the *Join*, *Prune*, *Assert*, and *Graft* messages. For brevity, we partially discuss our

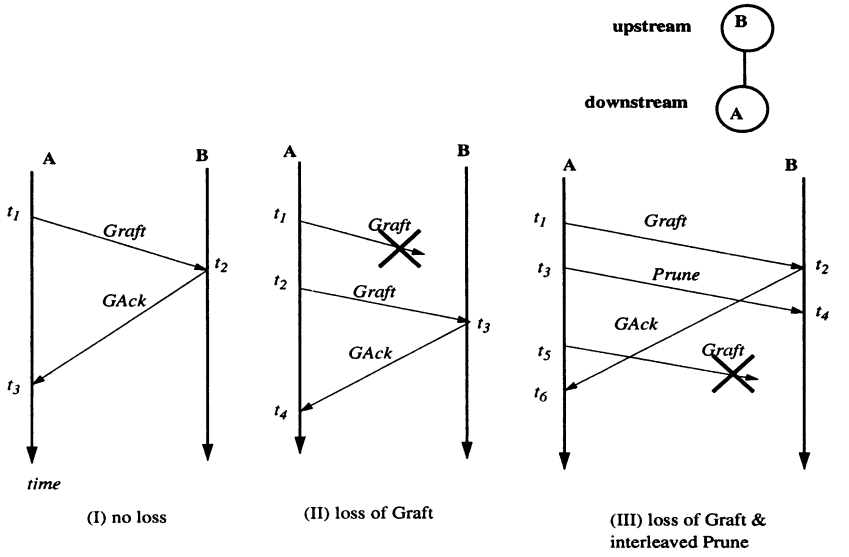

Figure 1.2 Graft event sequencing

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.

Join:. A scenario similar to that presented in section 1.4.5 incurred an error. In this case, the robustness violation was not allowing another chance to the downstream router to send a *Join*. A suggested fix would be to send another prune by F_{Del} before the timer expires.

Prune:. In the topology above, an error occurs when R_i loses the *Prune*, hence no *Join* is triggered. The fix suggested above takes care of this case too.

Assert:. An error in the *Assert* case occurs with no downstream routers; e.g. $G_I = \{F_i, F_j\}$. The design error is the absence of a mechanism to prevent pruning packets in this case. One suggested fix would be to have the *Assert* winner schedule a deletion timer (i.e. becomes F_{Del}) and have the downstream receiver (if any) send *Join* to the *Assert* winner.

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

Interleaving events and Sequencing. A *Graft* message is acknowledged by *GAck*, and is robust to message loss with the use of *Rtx* timer. Adversary external conditions are interleaved during the transient states and the *Rtx* timer is cleared, such that the adverse event will not be overridden by the *Rtx* mechanism.

To clear the *Rtx* timer, a transition should be created from NH_{Rtx} to NH which is triggered by a *GAck* according to the state dependency table ($NH \xleftarrow{GAck} NH_{Rtx}$). This transition is then inserted in the event sequence, and forward and backward implications are used to obtain the overall sequence of events illustrated in figure 1.2. In the first and second scenarios (**I** and **II**) no error occurs. In the third scenario (**III**) when a *Graft* followed by a *Prune* is interleaved with the *Graft* loss, the *Rtx* timer is reset with the receipt of the *GAck* for the first *Graft*, and the systems ends up in an error state. A suggested fix is to add sequence number to *Grafts*.

Loss of State. We consider momentary loss of state in a router. A '*Crash*' stimulus transfers any state 'X' into 'EU' or 'ED'. Hence, we add the following line to the transition table:

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

The FSM resumes function immediately after the crash (i.e. further transitions are not affected). 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* ($F_{Del} \xleftarrow{Prune} F$). Hence we may use the topologies constructed for *Prune* loss to analyze a crash for F_{Del} state.

Forward implication is then applied, and behavior after the crash is checked for correct packet delivery. To achieve this, host stimuli (i.e. *SPkt*, *HJ* and *L*) are applied, 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). The recovery is mainly due to the nature of PIM-DM; where protocol states are re-created with reception of data packets. This result is not likely to extend to protocols of other natures; e.g. PIM Sparse-Mode [14].

However, in violation with robustness requirements, there existed cases in which the system did not recover. In figure 1.3, the host joining in (II, a) did 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).

In figure 1.4 (II, a), the downstream router incurs join latency 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).

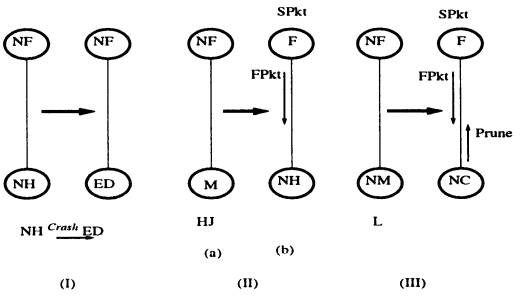

Figure 1.3 Crash leading to join latency

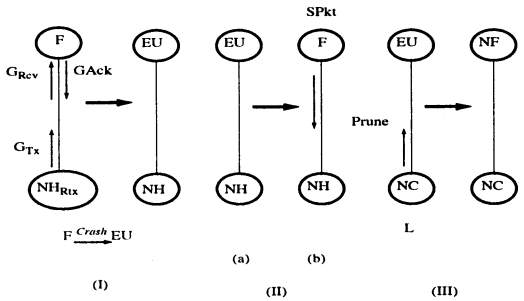

Figure 1.4 Crash leading to black holes

1.4.7 Limitations

The FOTG algorithms require a pre/post-condition global transition table, like the one presented in this paper. Generating such table from a conventional single router I/O FSM is part of our on-going work.

Currently the GFSM used in this study only models LANs. In our future work we will attempt to extend the model for regular and random topologies.

The LAN topologies constructed are inferred from the mechanisms specified by the transition table of the GFSM. The algorithm will not construct topologies resulting from non-specified mechanisms. For example, if the *Assert* mechanism was left out (due to a design error) the algorithm would not construct $\{F_i, F_j\}$ topology. So, FOTG (as presented here) may be used to evaluate behavior of specified mechanisms in the presence of network failures, but is not a general protocol verification tool.

1.5 SUMMARY AND FUTURE WORK

We have presented a new method for automating the robustness testing of multicast routing protocols, in the presence of network failures.

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 chip testing and FSM techniques, 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: using the state transition table, our method synthesizes $N - router$ LAN topologies necessary to generate protocol messages or states, in terms of a global system state.

Fault investigation: from the global state, forward implication is used to test the behavior of the system in the presence of faults.

Sequence of events: if an error is found, backward implication constructs a sequence of events leading to the erroneous state, which is used to analyze the protocol behavior.

Timing problems: we have presented an example of transforming a timing problem into a sequencing problem to analyze acknowledged messages.

We have conducted a case study for PIM-DM, and found several scenarios in which the protocol behaved erroneously.

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

We are in the process of conducting a quantitative analysis and evaluation of our method in terms of complexity and completeness; i.e. the number of topologies synthesized, state transitions traversed and faults covered. We are also investigating complexity reduction techniques by introducing equivalence classes of states and topologies, using counting equivalence and repetition constructors [11].

Future directions of this research include applying the FOTG method to other multicast protocols, including end-to-end performance analysis, extending the method to apply to topologies containing multiple LANs, and to include other network failures.

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] 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.

[6] 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.

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

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

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

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

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

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

[13] A. Helmy, D. Estrin, and S. Gupta. Fault-oriented Test Generation for Multicast Routing. *USC-CS-TR 98-673, www.usc.edu/dept/cs*, March 1998.

[14] 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.

Ahmed A-G Helmy received his M.S. ('95) from the University of Southern California, and his B.S. ('92) from Cairo University, Egypt, He is currently pursuing his Ph.D. in Computer Science at the University of Southern California. His research interests include protocol design and testing, multicast routing, and network simulation. Website: catarina.usc.edu/ahelmy.

Deborah Estrin is a Professor of Computer Science at the University of Southern California in Los Angeles where she joined the faculty in 1986. Estrin received her Ph.D. (1985) and M.S.(1982) from the Massachusetts Institute of Technology (M.I.T.) and her B.S. (1980) from U.C. Berkeley. In 1987, Estrin received the National Science Foundation, Presidential Young Investigator Award for her research in network interconnection and security. Estrin is a co-PI on the DARPA Virtual Internet Testbed (VINT) project and the NSF Routing Arbiter project at USC's Information Sciences Institute where she spends much of her time supervising doctoral student research.

Estrin is an active member of the IETF multicast routing related working groups and a long-time member of the End-to-End research group. Estrin is a member of the ACM, IEEE,and AAAS. She has served on numerous panels for The National Science Foundation, The National Academy of Engineering, and is currently a member of DARPA's Information Systems and Technology (ISAT) advisory board. Estrin has served as an editor for the ACM/IEEE Transaction on Networks and the Wiley Journal of Internetworking Research and Experience, and as a member of the program committee for many IEEE Infocom and ACM Sigcomm conferences, and she was program co-chair of ACM Sigcomm '96.

Sandeep Gupta received his Bachelors degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur, India in 1985 and obtained M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Massachusetts at Amherst in 1989 and 1991. Since 1991 he has been with the Department of Electrical Engineering – Systems at the University of Southern California, Los Angeles, where currently he is an Associate Professor. He is also a Co-Director of USC's M.S. Program in VLSI Design.

His research interests are in the area of VLSI Testing and Design and is currently involved in projects on Built-In Self-Test, delay testing and diagnosis of digital circuits, and test and validation of deep submicron, high speed circuits. He is a recipient of the National Science Foundation's Research Initiation Award (1992) and CAREER Award (1995).