FOTG: Fault-Oriented Stress Testing of IP Multicast

Ahmed Helmy, *Member, IEEE,* and Sandeep Gupta, *Member, IEEE*

***Abstract*— Network simulators provide a useful tool for protocol evaluation. However, the results depend heavily on the simulated scenarios, especially for complex protocols such as multicast. There has been little work on scenario generation. In this work we present a fault-oriented test generation (FOTG) algorithm for automated stress testing of multicast protocols. FOTG processes an extended FSM model, and uses a mix of forward and backward search techniques. Unlike traditional verification approaches, instead of starting from initial states, FOTG starts from a fault and uses cause-effect relations for automatic topology synthesis then uses backward implication to generate tests. Using FOTG we test various mechanisms commonly employed by multicast routing, and validate our results through simulation.**

***Index Terms*— Multicast routing, protocol testing.**

I. Introduction

NETWORK simulation [2] is valuable, but the results are only as good as the scenarios simulated. There is currently no systematic method to select simulation test scenarios, and most simulations are user-driven. Hence, there is a pressing need for scenario generation methods, especially for complex protocols such as multicast [1]. We present a framework for systematic testing of multicast protocols. In our experience, much of the protocol complexity lies in dealing with network failures. So, instead of using a *verification* approach, we use *falsification* of robustness, to expose protocol breaking points. Traditional verification approaches use reachability analysis [8] [9] [3] that employs forward search to inspect reachable states. Such approaches suffer from *state space explosion* problems. To ease this problem, state reduction [10] may be used. However, the main limitation of this approach is its inability to *synthesize topology*. We propose a fault-oriented test generation (*FOTG*) approach, where complete scenarios (including topology) are automatically generated. *FOTG* starts from a given fault and synthesizes the necessary conditions that trigger an error using forward and backward search techniques. *FOTG* borrows from principles of implication used in VLSI chip testing [11]. In VLSI, however, the topology is given, whereas for Internet protocols it must be synthesized, which adds a new dimension to our problem. We apply *FOTG* to analyze robustness of common mechanisms used in multicast routing, and reveal several of their design errors, even after years of deployment. Our method is applicable to similar classes of protocols.

Manuscript received May 15, 2004. This work was supported in part by Darpa, NSF, and NASA. The associate editor coordinating the review of this letter and approving it for publication was Dr. Nikos Nikolaou.

A. Helmy and S. Gupta are with the Electrical Engineering Department, University of Southern California, Los Angeles, CA 90089 (e-mail: helmy@usc.edu).

Digital Object Identifier 10.1109/LCOMM.2005.04021.

II. Multicast Routing Overview

Multicast routing delivers packets efficiently to group members by establishing trees via broadcast-and-prune or explicit join protocols. In broadcast-and-prune (DVMRP [1], PIM-DM [4]) multicast packets are broadcast. Leaf routers with no members send *Prune* message multicast hop-by-hop towards the source to stop further broadcasts. Routers with downstream members that receive a *Prune* send a *Join* message to maintain packet flow. Routers with new members send *Graft* messages to re-establish previously-pruned branches. *Grafts* are unicast hop-by-hop and acknowledged. In explicit join protocols, CBT [5], PIM-SM [6], SSM and Express [7], routers with members send *Join* messages multicast hop-by-hop toward the source or a root to build the multicast tree. Multicast protocols employ duplication/loop prevention; e.g., PIM protocols employ the *Assert* mechanism to elect at most one forwarder for each LAN. Other protocols use similar techniques. We target these common mechanistic building blocks (*Join*, *Prune*, *Graft*, *Assert*) and illustrate the results using PIM-DM.

III. Framework Overview

In contrast to average-case analysis using ad hoc scenarios, we focus on robustness analysis. Protocol robustness is the ability to operate correctly in presence of network failures. Our main contribution lies in developing new algorithms for automatic generation of scenarios (topology, event sequences, network failures) that drive protocols into undesirable states. Inputs to our method include **1)** protocol specification, as a global finite state machine (*GFSM*), and **2)** robustness definition using desirable conditions. The *GFSM* and desirable conditions are input to the test generation (*TG*) engine. This engine is the core of our method and includes algorithms for **i.** topology synthesis, **ii.** forward search and **iii.** backward search. The output of *TG* is a set of scenarios causing violation of desirable conditions. The scenarios include event sequences, network failures, and network topology. We initially model LAN topologies, then we extend our method to create more complex stress topologies.

A. Inputs: Protocol & Robustness Representation

We represent the protocol as a finite state machine (FSM), and the LAN topology as a global FSM (GFSM).

I. FSM model: A protocol running on a router i is modeled by a FSM $\mathcal{M}_i = (\mathcal{S}, \tau_i, \delta_i)$, where $\mathcal{S}$ is the set of states, τ_i is the set of stimuli, and $\delta_i : \mathcal{S} \times \tau_i \rightarrow \mathcal{S}$ is the state transition function describing the state transition rules. Following is our model of multicast routing. We define the states for router r_i on LAN l. (i) **System States** ($\mathcal{S}$): Table I shows possible router states. (ii) **Stimuli** (τ) include messages, timers and

TABLE I

POSSIBLE ROUTER STATES

State	Meaning	State	Meaning
F_i	Forwarder for the LAN	NR_i	Non-receiver
F_{i_Timer}	F_i with $Timer$ running	EU_i	Empty upstream
NF_i	Non-forwarder	ED_i	Empty downstream
R_i	Receiving packets from l	M_i	attached to member
R_{i_Timer}	R_i with $Timer$ running	NM_i	with no members

host events: **1.** Multicast messages include *Join*, *Prune*, *Assert*, forwarded packets *FPkt*. Unicast messages include *Graft* and Graft Ack (*GAck*) **2.** Timer events occur due to timer expiration (*Exp*) and include Graft resend timer (*Rs*), forwarder-deletion timer (*Del*), the events of their expiration (Rs_{Exp}, Del_{Exp}) and periodic timers. **3.** External host events (*Ext*) include sending packets (*SPkt*), join (*HJ*), and leave (*L*). Hence, τ ={*Join,Prune,Graft,GAck,Assert,FPkt,Rs,Del,SPkt,HJ,L*}.

II. GFSM model: The global state is a composition of router states. Outputs from one router may become inputs to others. Behavior of n routers on a LAN l is given by $\mathcal{M}_\mathcal{G} = (\mathcal{S}_\mathcal{G}, \tau_\mathcal{G}, \delta_\mathcal{G})$, where $\mathcal{S}_\mathcal{G} : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ is the global state space, $\tau_\mathcal{G} = \bigcup_{i=1}^{n} \tau_i$ is the set of stimuli, and $\delta_\mathcal{G}$ is the global state transition function $\mathcal{S}_\mathcal{G} \times \tau_\mathcal{G} \rightarrow \mathcal{S}_\mathcal{G}$. Example global state, G, is given by $\{F_1,R_2,NR_3,NR_4\}$ where r_1 is a forwarder, r_2 receiving from the LAN and r_3 and r_4 are non-receivers.

III. Fault model: A *fault* is a low level (e.g. physical layer) anomaly that may affect the protocol under test. Faults in our study include selective loss, where a multicast message is received by some routers but not others, router crash, and route inconsistency [12]. For brevity, we only present message loss. The design goal for many multicast routing protocols [6] is to be robust to single message loss events. For message loss, the transition due to the message is nullified.

The **transition table** describes, for each stimulus, the pre and post-conditions of its occurrence. A condition is given in terms of stimulus (*stim*), state (*state*) and transition (*trans*), where *trans* is given as *startState→endState*. A router may be (1) event originator, *orig*, (2) destination of message, *dst*, or (3) *other*. Table II shows transition table for PIM-DM.

• Pre-Conditions: may take the form **1.** *stim.state*, e.g., condition for *Join* message, $Prune_{other}.R_{orig}$; a *Prune* triggers a *Join* by a router in *R*, **2.** *stim.trans*, e.g., *HJ*.(*NR→R*); i.e. host join and transition from *NR* to *R*. If several pre-conditions exist, each triggers a stimulus.

• Post-Conditions: triggered by the stimulus and may take form: **1.** *trans*: has an implicit condition; *a→b* means if $a \in G$ then *a→b*, e.g. $NF_{dst} \rightarrow F_{dst}$. **2.** *condition.stim*: if *condition* then trigger *stim*, e.g. $R_{other}.Join_{other}$ means if $R_{other} \in G$ then $Join_{other}$. **3.** *stim.trans*, e.g., $GAck.(NF_{dst} \rightarrow F_{dst})$ means if $NF_{dst} \in G$ then transit to F_{dst} and trigger *GAck*.

Desirable conditions for multicast routing are to deliver data to members with least loss, latency, duplication and wastage. Hence, the conditions necessary to avoid undesirable behavior are: **1)** If one (or more) router is receiving from the LAN, then there must exist a forwarder. This avoids data loss, join latency or black holes. **2)** A LAN must have at most one forwarder at a time. This avoids packet duplication. **3)** If there exists a forwarder for the LAN, then there must be at least one router receiving from the LAN. This avoids wastage or leave latency. Thus, we identify the undesirable states as

TABLE II

TRANSITION TABLE FOR PIM-DM

(*Jn: Join, Pr: Prune, Asrt: Assert, Gr: Graft, o: other, d: dst*)

Stimulus	Pre-Conditions	Post-Conditions
Jn	$Pr_o.R_{orig}$	$F_{d\_Del} \rightarrow F_d$, $NF_d \rightarrow F_d$
Pr	*L.NR*, *FPkt.NR*	$F_d \rightarrow F_{d\_Del}$, $R_o.Jn_o$
Gr	*HJ*.($NR \rightarrow R_{\_Rs}$), Rs_{Exp}	$GAck.(NF_d \rightarrow F_d)$
GAck	$Gr_o.(NF_{orig} \rightarrow F_{orig})$	$R_{d\_Rs} \rightarrow R_d$
Asrt	$FPkt_o.F_{orig}$	$F_o \rightarrow NF_o$
FPkt	*SPkt.F*	*Pr*.(*NM→NR*), *ED→R*, *M→R*, $EU_o \rightarrow F_o$, $F_o.Ast$
Rs	Rs_{Exp}	*Gr*
Del	Del_{Exp}	$F_{orig\_Del} \rightarrow NF_{orig}$
SPkt	*Ext*	$FPkt.(EU_{orig} \rightarrow F_{orig})$
HJ	*Ext*	*NM→M*, *Gr*.($NR \rightarrow R_{\_Rs}$)
L	*Ext*	*M→NM*, *Pr*.(*R→NR*), *Pr*.($R_{\_Rs} \rightarrow NR$)

$\{F_i,F_j,\ldots\}$, $\{R_i,\ldots,\{X_j\text{-}F_j\}\}$, and $\{F_i,\ldots,\{X_j\text{-}R_j\}\}$. The term $\{X_j\text{-}F_j\}$ denotes any state except forwarding, and $\{X_j\text{-}R_j\}$ is any state except *R*. We distinguish between *transient* and *stable* states and identify *externally triggered* (*ETT*) and *internally triggered* transitions (*ITT*). A global state is checked at the end of *ETT* after completing its dependent *ITTs*.

B. The Output Test Scenarios

An output test scenario includes sequences of host events, network faults and LAN/WAN topology that cause the protocol to violate a desirable condition. Host events include *join*, *leave*, or *send*. Network faults include losses, crashes or inconsistent routes. The topology consists of network layer multicast nodes.

IV. FAULT-ORIENTED TEST GENERATION (FOTG)

The *FOTG* algorithm has three main stages: a) sub-topology synthesis establishes states on a LAN necessary to trigger the target message. This forms a global state, G_I, in the middle of the state space, b) forward search is then performed from G_I after applying the fault. This is called *forward implication*. Succeeding stable state is checked for errors, c) if an error occurs, backward search is performed to establish a sequence leading from an initial state (*I.S.*) to G_I. This process is called *backward implication*. The algorithmic details are based on *condition→effect* reasoning of the transition rules.

Sub-Topology Synthesis starts from a protocol message and uses the transition table to synthesize G_I to satisfy the conditions to trigger and get affected by this message as follows: **1.** Initially G_I is empty and the inspected stimulus (*IStim*) is set to the given protocol message. **2.** For *IStim*, *startState*(s) of the post-conditions and *endState*(s) of the pre-conditions are obtained. If these states do not exist in G_I, and cannot be inferred therefrom, then they are added to G_I. **3.** Get stimulus of the pre-condition of *IStim*, call it *newStim*. If *newStim* is not external then set *IStim* to *newStim* and go to 2.

Forward Implication obtains G_{I+1} by applying the transitions rules, timer expiration and loss, starting from G_I. If loss affects more than one state, then the space is expanded to include all selective loss scenarios for the affected routers.

Backward Implication obtains sequence of events leading to G_I from *I.S.* For states in G_I possible backward implications are applied to obtain backward steps. This is repeated, depth first, for preceding states. If all backward branches are exhausted and no *I.S.* is reached then G_I is unreachable. To

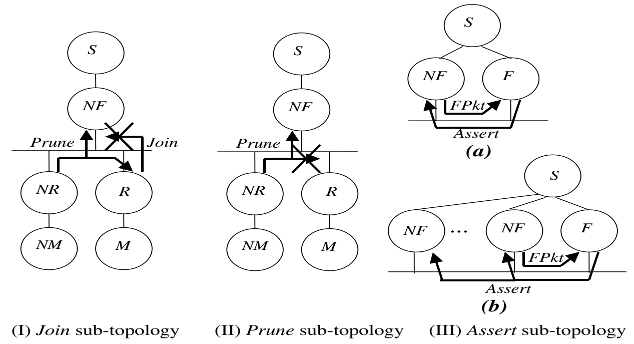

Fig. 1. Synthesized sub-topologies for Join, Prune and Assert.

rewind the global state one step backward, transition rules are reversed. For unicast messages the state of the destination is rolled back, but for multicast all affected states are rolled back. We ensure for every backward step there is a corresponding valid forward step. Upon contradiction we *backtrack*.

Topology Expansion uses the *sub-topologies* synthesized thus far as *building blocks* to form complex topologies. To avoid *error hiding*, where errors in one sub-topology hide errors in another, two conditions must be met: (1) stimuli propagation, and (2) error isolation. Stimuli are triggered by data packets or host events. To propagate the packets, we construct disjoint paths from the source to all sub-topologies. Also, we attach hosts as needed to trigger host events. To isolate errors, we use error-free fork points to the sub-topologies.

V. Applying the Method

We apply our method to study selective loss of *Join*, *Prune*, *Assert* and *Graft*. For a *Join* loss example, with *I.S.*=$\{NM, EU\}$, the following steps are taken:

Synthesizing the Global State

1. *Join*: *startState* of post-condition is $NF_{dst} \Rightarrow G_I=\{NF_k\}$
2. *Join*: state of pre-condition is $R_i \Rightarrow G_I=\{R_i, NF_k\}$, goto *Prune*
3. *Prune*: *startState* of post-condition is F_k, implied from NF_k in G_I
4. *Prune*: state of pre-condition is $NR_j \Rightarrow G_I=\{R_i, NF_k, NR_j\}$, goto *L* (*Ext*)
5. *startState* of post-condition is *R* can be implied from *NR* in G_I

Forward Implication

loss w.r.t. r_j: $G_I=\{R_i, NF_k, NR_j\} \rightarrow G_{I+1}=\{R_i, NF_k, NR_j\}$ error

Backward implication

$G_I=\{R_i, NF_k, NR_j\} \overset{Prune}{\longleftarrow} G_{I-1}=\{R_i, F_k, NR_j\} \overset{FPkt}{\longleftarrow} G_{I-2}=\{M_i, F_k, NM_j\}$
$\overset{SPkt}{\longleftarrow} G_{I-3}=\{M_i, EU_k, NM_j\} \overset{HJ_i}{\longleftarrow} G_{I-4}=\{NM_i, EU_k, NM_j\}$=*I.S.*

When router r_k loses the Join an error state occurs where router r_i is expecting packets, with no forwarder for LAN l.

A. Summary of Results

We present results for *Join*, *Prune*, *Assert* and *Graft*.

Join/ Prune: in Fig. 1 (I) and in Fig. 1 (II) an error occurs when R_i loses the *Prune*, and no *Join* is sent. To fix these errors a *Prune* is sent by F_{Del} before timer expires.

Assert: in Fig. 1 (III) bandwidth wastage error occurs with no downstream routers; e.g. $G_I=\{F_i, F_j\}$, where the packets are never pruned. To fix, the *Assert* winner schedules a deletion (F_{Del}) and downstream receivers (if any) *Join* to that winner.

Graft: if a host performs *HJ,L,HJ* in a short period and the 2nd *Graft* is lost, an error state occurs due to *Rs* reset by the first *GAck*. To fix we add sequence numbers to *Grafts*.

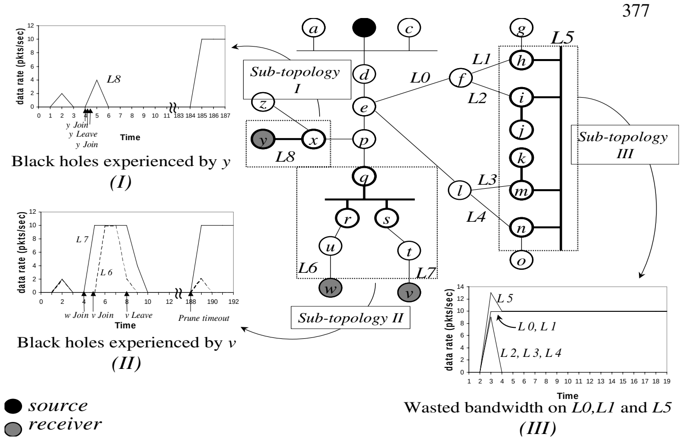

Fig. 2. The simulated topology includes several sub-topologies automatically synthesized by STRESS. The overall topology experienced black holes and bandwidth wastage as was predicted by our method. The graphs show packet rates over specific links to illustrate the protocol errors. The STRESS sub-topologies are shown in bold.

B. Simulations of Extended Synthesized Scenarios

To validate our scenarios, we conduct NS-2 [2] simulations. We use topology expansion to get the example 26-node stress topology in Fig. 2. The figure shows black holes and bandwidth wastage errors as predicted by our method. After applying the fixes proposed in this study, these errors were eliminated. This was validated by running *FOTG* and NS simulations. The complexity for $FOTG$ was quite manageable for all our case studies. These corrections were integrated into PIM-DM/SM specifications.

In conclusion, we have introduced the *FOTG* algorithm for multicast testing. Strength of *FOTG* comes from its ability to construct error scenarios and topologies by starting directly from the faults, hence it is best fit for robustness studies. Our algorithm operates on transition table entries and hence applies to other classes of protocols that use similar semantics and mechanisms.

References

[1] D. Waitzman, S. Deering, and C. Partridge, "Distance vector multicast routing protocol," RFC1075, Nov. 1988.
[2] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, *et al.*, "Advances in network simulation," *IEEE Computer*, vol. 33, pp. 59-67, May 2000.
[3] F. Lin, P. Chu, and M. Liu, "Protocol verification using reachability analysis," *Computer Communications Review*, vol. 17, pp. 126-135, Aug. 1987.
[4] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei, "Protocol independent multicast - dense mode: protocol specification," proposed IETF RFC, Sept. 96.
[5] A. Ballardie, P. Francis, and J. Crowcroft, "Core based trees," *ACM Sigcomm*, vol. 23, pp. 85-95, Sept. 1993.
[6] D. Estrin, D. Farinacci, A. Helmy, *et al.*, "Protocol independent multicast - sparse mode (PIM-SM): motivation and architecture," proposed IETF RFC, Oct. 1996.
[7] H. Holbrook and D. Cheriton, "EXPRESS support for large-scale single-source applications," *ACM SIGCOMM*, vol. 29, pp. 65-78, Sept. 1999.
[8] E. Clarke and J. Wing, "Formal Methods: State of the Art and Future Directions," in *Proc. ACM Workshop on Strategic Directions in Computing Research*, pp 626-643, Dec. 1996.
[9] C. Jones, *Systematic Software Development using VDM*. Englewood Cliffs, NJ: Prentice-Hall, 1990.
[10] G. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[11] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. AT&T Labs, 1990.
[12] A. Helmy, "Systematic test synthesis for multipoint protocol design," USC-CS-TR-99-716, Ph.D. dissertation, Aug. 1999.