Systematic Testing of Multicast Routing Protocols: Analysis of Forward and Backward Search Techniques

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

Abstract—

In this paper, we present a new methodology for developing systematic and automatic test generation algorithms for multipoint protocols. These algorithms attempt to synthesize network topologies and sequences of events that stress the protocol's correctness or performance. This problem can be viewed as a domain-specific search problem that suffers from the state space explosion problem. One goal of this work is to circumvent the state space explosion problem utilizing knowledge of network and fault modeling, and multipoint protocols. The two approaches investigated in this study are based on forward and backward search techniques. We use an extended finite state machine (FSM) model of the protocol. The first algorithm uses forward search to perform reduced reachability analysis. Using domain-specific information for multicast routing over LANs, the algorithm complexity is reduced from exponential to polynomial in the number of routers. This approach, however, does not fully automate topology synthesis. The second algorithm, the fault-oriented test generation, uses backward search for topology synthesis and uses backtracking to generate event sequences instead of searching forward from initial states. Using these algorithms, we have conducted studies for correctness of the multicast routing protocol PIM.

I. Introduction

Network protocols are becoming more complex with the exponential growth of the Internet, and the introduction of new network services. In particular, the advent of IP multicast and the MBone enabled multipoint applications. To date, little effort has been exerted to formulate systematic methods and tools that aid in the design and characterization of these protocols.

In addition, researchers are observing new and obscure, yet all too frequent, failure modes over the internets [1]. Such failures are becoming more frequent, mainly due to increased heterogeneity, and may affect the proper operation of network protocols.

Network protocol errors are often detected by application failure. Such errors are hardest to diagnose when the behavior is unexpected. Even if a protocol is proven to be correct in isolation, its behavior may be unpredictable in an operational network, with interaction with other protocols and the presence of network failures. To provide a systematic methodology for testing of multipoint protocols, we present the STRESS framework that integrates test generation algorithms with simulation and implementation. We target robustness testing in the presence of network failures.

We investigate two approaches for test generation. The first approach, called the fault-independent test generation (FITG), uses a forward search algorithm to explore a subset of the protocol state space to generate the test events automatically. State and fault equivalence relations are used in this approach to reduce the search complexity. The second approach is called the fault-oriented test generation (FOTG), and uses a mix of forward and backward search techniques to synthesize test events and topologies automatically. We have applied these methods to multicast routing. Our case studies revealed several design errors, for which we have formulated solutions with the aid of our systematic method.

The rest of this document is organized as follows. Section II introduces the STRESS framework. Sections III, IV, V present the search techniques, FITG and FOTG, respectively. Section VI presents related work and Section VII concludes.

II. Framework Overview

Robustness of a protocol is its ability to respond correctly in the face of network component failures and packet loss. This work presents a methodology for studying and evaluating multicast protocols, specifically addressing robustness and performance issues. We propose a framework that integrates automatic test generation as a basic component for protocol design, along with protocol modeling, simulation and implementation testing. The major contribution of this work lies in developing new methods for generating stress test scenarios that target robustness and correctness violation.

Our framework integrates test generation with simulation and implementation code. We call our proposal **S***ystematic* **T***esting of* **R***obustness by* **E***valuation of* **S***ynthesized* **S***cenarios (STRESS).* As the name implies, systematic methods for scenario synthesis are a core part of the framework. We use the term scenarios to denote the test-suite consisting of the topology and events.

The input to this framework is the specification of a protocol, and a definition of its correctness requirements. Usually robustness is defined in terms of network dynamics or fault models. A fault model represents various component faults; such as packet loss, or machine crashes. The desired output is a set of test-suites that stress the protocol mechanisms according to the robustness criteria.

As shown in Figure 1, the STRESS framework includes test generation, detailed simulation driven by the synthesized tests, and protocol implementation driven through an emulation interface to the simulator. Here, we focus on the test generation component.

A. Test Generation

The core contribution of our work lies in the development of systematic test generation algorithms for protocol robustness. Automatic test generation (TG) produces tests based on a model of the protocol. TG can be: a) fault-independent, or b) fault-oriented. Fault-independent TG 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. We use the notion of equivalence to reduce the search complexity.

In contrast, fault-oriented tests are generated for specified faults. Fault-oriented test generation starts from the fault (e.g. a lost message) and synthesizes the necessary topology and sequence of events that trigger the error. This algorithm uses a mix of forward and backward searches. We conduct case studies for the multicast routing protocol PIM-DM to illustrate differences between the approaches, and provide a basis for their comparison.

B. The system model

We define our target system in terms of network and topology elements and a fault model. Elements of the network consist of multicast capable nodes and symmetric links. The topology is an N-router LAN. A *fault* is a low level (e.g., physical layer) anomalous

0-7803-6494-5/00/$10.00 © 2000 IEEE

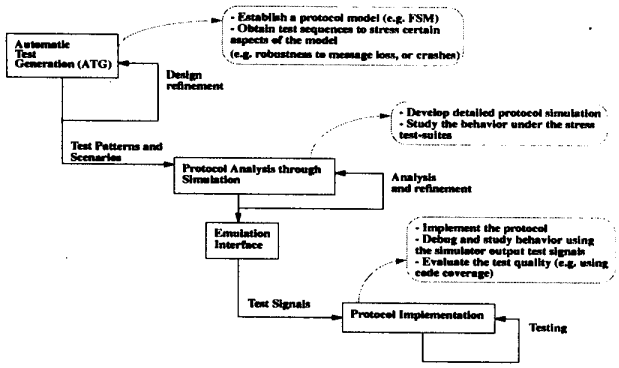

Fig. 1. The STRESS framework

behavior, that may affect the behavior of the protocol under test [1]. The fault model may include: (a) Packet loss due to congestion or failures. We consider selective packet loss, where a multicast packet is received by some group members but not others, (b) Loss of state, e.g., multicast and/or unicast routing tables, due to crashes, and (c) Network delays.

Usually, a fault model is defined in conjunction with the robustness criteria for the protocol. For our robustness studies we study PIM. One design goal for PIM is to be able to recover gracefully from single protocol message loss [2]. In addition, we study PIM protocol behavior in presence of crashes.

C. Test Sequence Definition

For our robustness studies we adopt a single-fault model, where a single fault occurs during a test sequence. We define two 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 the stable state $P(q, T')$ reached after the occurrence of the fault is incorrect [3].

D. Test Scenario

A test scenario is defined by a sequence of (host) events, a topology, and a fault model. The events are actions performed by the host and act as input to the system; e.g., join, leave, or send packet. The topology is the routed topology of set of nodes and links. The nodes run the set of protocols under test or other supporting protocols. The links can be either point-to-point links or LANs. The fault model is used to inject the fault into the test [4].

E. Brief description of PIM-DM

We apply our automatic test generation algorithms to a version of the Protocol Independent Multicast-Dense Mode, or PIM-DM [2] [5]. PIM-DM uses broadcast-and-prune to establish the multicast distribution trees. 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 get 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 prevents these situations and ensures there is at most one forwarder for a LAN.

The correct function of a multicast routing protocol is to deliver data from senders to group members without data loss. For our methods, we assume that a correctness definition is given by the protocol specification. For illustration, we discuss the protocol errors and the correctness conditions.

PIM Protocol Errors and Correctness Conditions. In this study we target protocol design and specification errors. We are interested mainly in erroneous stable (i.e. non-transient) states. In general, the protocol errors may be defined in terms of the end-to-end behavior as functional correctness requirements. In our case, for PIM-DM, an 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*: lack of packet delivery after a receiver joins the group, 5) *leave latency*: unnecessary packet delivery after a receiver leaves the group [6], and 6) *wasted bandwidth*: unnecessary packet delivery to network links that do not lead to group members.

We assume that correctness conditions are provided by the protocol designer or the protocol specification. These conditions are necessary to avoid the above protocol errors in a LAN environment, and include: 1) If one (or more) of the routers is expecting to receive packets from the LAN, then one other router must be a forwarder for the LAN. Violation of this condition may lead to data loss (e.g. join latency or black holes). 2) The LAN must have at most one forwarder at a time. Violation of this condition may lead to data packet duplication. 3) If one of the routers is a forwarder for the LAN, then there must be at least one router expecting packets from the LANs. Violation of this condition may lead to leave latency.

[1] We distinguish between the terms *error* and *fault*. An *error* is a failure of the protocol as defined in the protocol specification. For example, duplication in packet delivery is an error for multicast routing. Note that a fault may not necessarily be an error for the low level protocol.

[2] That is, being robust to a single message loss implies that transitions cause the protocol to move from one correct stable state to another, even in the presence of selective message loss.

[3] 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. We are only concerned with the stable (i.e. non-transient) behavior of a protocol.

[4] According to our single-message loss model, for example, a fault may denote the 'loss of the second message of type $prune$ traversing a certain link'. Knowing the location and the triggering action of the fault is important in analyzing the protocol behavior.

[5] 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. When routers are connected via a LAN, hop-by-hop messages are multicast on the LAN, and may experience selective loss; i.e. may be received by some nodes but not others.

[6] Join and leave latencies may be considered in other contexts as performance issues. However, in our study we treat them as errors.

III. Search Techniques

The problem of test synthesis can be viewed as a search problem. By searching the possible sequences of events and faults over network topologies and checking for design requirements, we can construct the test scenarios that stress the protocol. However, due to the state space explosion, techniques must be used to reduce the complexity of the space to be searched. We attempt to use these techniques to achieve high test quality and protocol coverage.

Following we present the GFSM model for the case study protocol (PIM-DM) to use it to analyze the complexity of the search and illustrate the algorithmic principles for FITG and FOTG.

A. The Protocol Model

We represent the protocol as a finite state machine (FSM) and the overall LAN system by a global FSM (GFSM).

I. FSM model: Every instance of the protocol is modeled by a deterministic FSM consisting of: (i) a set of states, (ii) a set of stimuli causing state transitions, and (iii) state transition rules. For a system i, this is represented by the machine $\mathcal{M}_i = (\mathcal{S}, \tau_i, \delta_i)$, where $\mathcal{S}$ is a finite set of state symbols, τ_i is the set of stimuli, and δ_i is the state transition function $\mathcal{S} \times \tau_i \rightarrow \mathcal{S}$.

II. Global FSM model: The global state is defined as the composition of individual router states. The output messages from one router may become input messages to other routers. Such interaction is captured by the GFSM model in the global transition table. The behavior of a system with n routers may be described by $\mathcal{M}_\mathcal{G} = (\mathcal{S}_\mathcal{G}, \tau_\mathcal{G}, \delta_\mathcal{G})$, where $\mathcal{S}_\mathcal{G}$: $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ 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 $\mathcal{S}_\mathcal{G} \times \tau_\mathcal{G} \rightarrow \mathcal{S}_\mathcal{G}$.

The fault model is integrated into the GFSM model. For message loss, the transition caused by the message is nullified. Crashes may be treated as stimuli causing the routers affected by the crash to transit into a *crashed* state.

B. PIM-DM Model

FSM model $\mathcal{M}_i = (\mathcal{S}_i, \tau_i, \delta_i)$. For a given group and a given source (i.e., 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 a receiver expecting packets from) the LAN.

System States ($\mathcal{S}$).

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 a non-forwarder
NH_i	Router i has the LAN as its next-hop
NH_{i_Timer}	same as NH_i with Timer $_{Timer}$ running
NC_i	Router i has a negative-cache entry
EU_i	Upstream router i is empty
ED_i	Downstream router i is empty
M_i	Downstream router with attached member
NM_i	Downstream router with no members

Possible states for *upstream* and *downstream* routers are:

$$\mathcal{S}_i = \begin{cases} \{F_i, F_{i\_Timer}, NF_i, EU_i\}, & \text{upstream;} \\ \{NH_i, NH_{i\_Timer}, NC_i, M_i, NM_i, ED_i\}, & \text{o.w..} \end{cases}$$

Stimuli (τ). 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 Ack reception $GAck$, Assert reception $Assert$, 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 $DelExp$. We refer to the event of timer expiration as ($TimerImplication$).
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.

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

Global FSM (GFSM) model. Subscripts are added to distinguish different routers. These subscripts are used to describe router semantics and how routers interact on a LAN. An example global state for a topology of 4 routers connected to a LAN, with router 1 as a forwarder, 2 expecting packets, and 3 and 4 have negative caches, is given by $\{F_1, NH_2, NC_3, NC_4\}$ [7].

C. Defining stable states

We are concerned with stable state (i.e. non-transient) behavior, defined in this section. 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. 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 $Graft$) [8].

A global state is checked for correctness at the end of an ET transition after completing its dependent IT transitions. Following is the table of events used.

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

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 temporarily seem to 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

[7] See Section V for more detailed semantics of GFSM.

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

stable state. Event-triggered timers (e.g., *Del*, *Rtx*) fire at the end of a complete transition.

IV. Fault-independent Test Generation

Fault-independent test generation (FITG) uses the forward search technique to investigate parts of the state space. As in reachability analysis, forward search starts from initial states and applies the stimuli repeatedly to produce the reachable state space (or part thereof). Conventionally, an exhaustive search is conducted to explore the state space. In the exhaustive approach all reachable states are expanded until the reachable state space is exhausted. We use several manifestations of the notion of equivalence to reduce the complexity of the exhaustive algorithm and expand only equivalent subspaces. To examine robustness of the protocol, we incorporate selective loss scenarios into the search.

A. Reduction Using Equivalences

The search procedure starts from the initial states [9] and keeps a list of states visited to prevent looping. Each state is expanded by applying the stimuli and advancing the state machine forward by implementing the transition rules and returning a new stable state each time. We use the equivalence notion to reduce the complexity of the search in three stages of the search. The first reduction we use is to investigate only the equivalent initial states. To achieve this we simply treat the set of states constituting the global state as unordered set instead of ordered set. For example, the output of such procedure for $I.S. = \{NM, EU\}$ and the number of routers $n = 2$ would be: $\{NM, NM\}, \{NM, EU\}, \{EU, EU\}$. The second reduction we use is during comparison of visited states. Instead of comparing the actual states, we compare and store equivalent states. Hence, for example, the states $\{NF_1, NH_2\}$ and $\{NH_1, NF_2\}$ are equivalent. The third reduction is made based on the observation that applying identical stimuli to different routers in identical states leads to equivalent global states. Hence, we can eliminate some redundant transitions. For example, for the global state $\{NH_1, NH_2, F_3\}$ a *Leave* applied to R_1 or R_2 would produce the equivalent state $\{NH^1, NC^1, F^1\}$. We call the algorithm after the third reduction the **reduced** algorithm. In all the above algorithms, a forward step advances the GFSM to the next stable state. This is done by applying all the internally dependent stimuli (elicited due to the applied external stimulus) in addition to any timer implications, if any exists. Only stable states are checked for correctness.

[9] For our case study the routers start as either a non-member (NM) or empty upstream routers (EU), that is, the initial states $I.S. = \{NM, EU\}$.

	Expanded States		Forwards	
Rtrs	Exhaustive	Reduced	Exhaustive	Reduced
3	178	30	2840	263
4	644	48	14385	503
6	7480	106	271019	1430
8	80830	200	4122729	3189
10	843440	338	55951533	6092
12	8621630	528	708071468	10483
14	86885238	778	8.546E+09	16738
	Transitions		Errors	
Rtrs	Exhaustive	Reduced	Exhaustive	Reduced
3	343	65	33	6
4	1293	119	191	13
6	14962	307	3235	43
8	158913	633	41977	101
10	1638871	1133	491195	195
12	16666549	1843	5441177	333
14	167757882	2799	58220193	523

Fig. 2. Simulation statistics for forward search.

B. Applying the Method

The protocol model is provided by the protocol specification, in terms of transition rules of the GFSM, and a set of initial state symbols. The design requirements for correctness are assumed to be also given by the protocol specification. This includes definition of correct states or erroneous states, in addition to the fault model if studying robustness. Also, the number of routers in the topology or topologies to be investigated (i.e., on the LAN) has to be specified.

Complexity of forward search for PIM-DM. We identified the initial state symbols to be $\{NM, EU\}$; NM for downstream routers and EU for upstream routers. The number of reachable states visited, the number of transitions and the number of erroneous states found were recorded. Summary of the results is given in Figure 2. The number of expanded states denotes the number of visited stable states. The number of 'forwards' is the number of times the state machine was advanced forward denoting the number of transitions between stable states. The number of transitions is the number of visited transient states, and the number of error states is the number of stable (or expanded) states violating the correctness conditions [10]. We notice significant reduction in the algorithm complexity with the use of equivalence relations. The number of transitions is reduced from $O(4^n)$ for the exhaustive algorithm, to $O(n^4)$ for the **reduced** algorithm. Similar results were obtained for the number of forwards, expanded states and number of error states. The reduction gained by using the equivalence is exponential [11].

Summary of behavioral errors for PIM-DM. We used the above algorithm to search the protocol model for PIM-DM. Correctness was checked automatically by the method by checking the stable states. By analyzing the sequence of events leading to error we were able to reason about the protocol behavior. We have studied

[10] Note that each of the other error states is equivalent to at least one error state detected by the **reduced** algorithm. Hence, having less number of discovered error states by an algorithm in this case does not mean losing any information or causes of error, which follows from the definition of equivalence. Reducing the error states means reducing the time needed to analyze the errors.

[11] More detailed presentation of the algorithmic details and results are given in [3].

cases of up to 14-router LANs. More than 6 errors were discovered, causing wasted bandwidth, and black holes due to selective loss of $Prune$ or $Join$ messages.

Limitations. We should note some limitations of the current FITG method. The topology is an input to the method in terms of number of routers [12]. Equivalence classes are currently given as input to the method. In this study we have used symmetries inherent in multicast routing on LANs. Identification of other equivalence classes is part of future work. The topology used in this study is limited to a single-hop LAN. Our work in [4] introduces the notion of *virtual* LAN to represent the multicast distribution tree.

In sum, the fault-independent test generation may be used for protocol verification given the symmetry inherent in the system studied (i.e., protocol and topology). For robustness studies, where the fault model is included in the search, the complexity of the search grows.

V. Fault-oriented Test Generation

In the fault-oriented test generation (FOTG) method, the tests are generated for specific faults. The test generation algorithm starts from the fault(s) and searches for a possible error, establishing the necessary topology and events to produce the error. Once the error is established, a backward search technique produces a test sequence leading to the erroneous state, if such a state is reachable.

A. FOTG Method Overview

Fault-oriented test generation (FOTG) targets specific faults or conditions, and so is better suited to study robustness in the presence of faults in general. FOTG has three main stages: a) topology synthesis, b) forward implication and error detection, and c) backward implication. The topology synthesis establishes the necessary components (e.g., routers and hosts) of the system to trigger the given condition (e.g., trigger a protocol message). This leads to the formation of a global state in the middle of the state space [13]. Forward search is then performed from that global state in its vicinity, i.e., within a complete transition, after applying the fault. This process is called *forward implication*, and uses the search techniques in Section IV. If an error occurs, backward search is performed thereafter to establish a valid sequence leading from an initial state to the synthesized global state. To achieve this, the transition rules are reversed and a search is performed until an initial state is reached, or the synthesized state is declared unreachable. This process is called *backward implication*. The algorithmic details are mainly based on $condition \rightarrow effect$ reasoning of the transition rules. This reasoning is emphasized in the semantics of the transition table.

[12] In the next section we present a new method that synthesizes the topology automatically as part of the search process.

[13] The global state from which FOTG starts is synthesized for a given fault, such as a message to be lost.

B. The Transition Table

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 $startState \rightarrow endState$. We further extend message and router semantics to capture multicast semantics.

- Stimuli and router semantics: Stimuli are classified based on the routers affected by them. Stimuli types include events that affect only the originating router and include HJ, L, $SPkt$, $Graft_{Tx}$, Del and Rtx, unicast messages that are only processed by the destination, including $GAck$ and $Graft_{Rcv}$, and multicast messages processed by all other routers in the group, including $Assert$, $Join$, $Prune$ and $FPkt$. According to these different types of stimuli processing a router may take as subscript '$orig$', 'dst', or '$other$'. The '$orig$' symbol designates the originating router of the stimulus or message, whereas 'dst' designates the destination of the message. '$other$' indicates routers other than the originator.
- Pre-Conditions: are of the form $stimulus.state/transition$, where the transition is given as $startState \rightarrow endState$. If there are several pre-conditions, at least one pre-condition is necessary to trigger the stimulus. Example of a $stimulus.state$ condition is the condition for $Join$ message, namely, $Prune_{other}.NH_{orig}$, that is, a $Join$ is triggered by the reception of a $Prune$ from another router, with the originator of the $Join$ in NH. An example of a $stimulus.transition$ condition is the condition for Graft transmission $HJ.(NC \rightarrow NH)$; i.e. a host joining and the transition of the router from the negative cache state to the next hop state.
- Post-Conditions: A post-condition is an event and/or transition that is triggered by the stimulus. Post-conditions may be in the form: 1. $transition$: has an implicit condition; i.e. '$a \rightarrow b$' means 'if $a \in GState$ then $a \rightarrow b$', e.g. the $Join$ post-condition ($NF_{dst} \rightarrow F_{dst}$). 2. $Condition.stimulus$: if the condition is satisfied then the stimulus is triggered, e.g. $Prune$ post-condition '$NH_{other}.Join_{other}$' means that if $NH_{other} \in GState$ then $Join_{other}$. 3. $Stimulus.transition$: has the transition condition implied as in (1) above. For example, $Graft_{Rcv}$ post-condition '$GAck.(NF_{dst} \rightarrow F_{dst})$', means if $NF_{dst} \in GState$, then the transition occurs and $GAck$ is triggered [14].

[14] If more than one post-condition exists, then the logical relation between them is either an 'XOR' if the router is the same, or an 'AND' if the routers are different. For example, $Join$ post-conditions are '$F_{dst\_Del} \rightarrow F_{dst}, NF_{dst} \rightarrow F_{dst}$', which means ($F_{dst\_Del} \rightarrow F_{dst}$) XOR ($NF_{dst} \rightarrow F_{dst}$). On the other hand, $Prune$ post-conditions are '$F_{dst} \rightarrow F_{dst\_Del}, NH_{other}.Join_{other}$', which implies that the transition will occur if $F_{dst} \in GState$ AND a $Join$ will be triggered if $NH \in GState$.

Following is a partial transition table used in our case study.

Stimulus	Pre-conditions	Post-conditions
$Join$	$Prune_{other}.NH_{orig}$	$F_{dst\_Del} \rightarrow F_{dst}, NF_{dst} \rightarrow F_{dst}$
$Prune$	$L.NC, FPkt.NC$	$F_{dst} \rightarrow F_{dst\_Del}$, $NH_{other}.Join_{other}$
$Graft_{Tx}$	$HJ.(NC \rightarrow NH)$, $RtxExp.(NH_{\_Rtx} \rightarrow NH)$	$Graft_{Rcv}.(NH \rightarrow NH_{\_Rtx})$
$Graft_{Rcv}$	$Graft_{Tx}(NH \rightarrow NH_{\_Rtx})$	$GAck.(NF_{dst} \rightarrow F_{dst})$
$GAck$	$Graft_{Rcv}.F$	$NH_{dst\_Rtx} \rightarrow NH_{dst}$
$HJoin$	Ext	$NM \rightarrow M, Graft_{Tx}.(NC \rightarrow NH)$

C. FOTG details

Our FOTG approach consists of three phases: synthesizing the global state, forward implication, and backward implication.

Synthesizing the Global State. Starting from a condition (e.g., protocol message) and using the transition table, a global state is synthesized. We refer to this state as the global-state inspected (G_I), and it is obtained as follows:

1. The global state is initially empty and the inspected stimulus is initially set to the stimulus investigated.
2. For the inspected stimulus, the $startState$(s) of the transition of the post-condition are obtained. If these states do not exist in the global state G, and cannot be inferred therefrom, then they are added to the global state.
3. For the inspected stimulus, the $endState$(s) of the transition of the pre-condition are obtained. If these states do not exist in, and cannot be inferred from G then they are added to the global state.
4. Get stimulus of the pre-condition of the inspected stimulus, call it $newStim$. If $newStim$ is not external (Ext), then set the inspected stimulus to $newStim$, and go back to step 2 [15].

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

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). Furthermore, faults are incorporated into the search. For example, in the case of a message loss, the transition that would have resulted from the message is not applied. If more than one state is affected by the message, then the space is expanded to include the various selective loss scenarios for the affected routers. For crashes, the routers affected by the crash transit into the crashed state as defined by the expanded transition rules. Forward implication uses the forward search techniques described earlier in Section IV [16].

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

Backward steps are taken for the components in the global state G_I, each step producing another global state $GState$. For each state in $GState$ possible backward implication rules are attempted to obtain valid backward steps toward an initial state. This process is repeated for preceding states in a depth first fashion. If all backward branches are exhausted and no initial state was reached the state is declared unreachable.

To rewind the global state one step backward, the reverse transition rules are applied. Depending on the stimulus type of the backward rule, different states in $GState$ are rolled back, e.g., for unicast destination of the stimulus is rolled back, but for multicast, all affected states are rolled back.

Note, however, that not all backward steps are valid, and backtracking is performed when a backward step is invalid. Backtracking may occur when the preceding states contradict the rules of the protocol. These contradictions may manifest themselves as:

- Src not found: src is the originator of the stimulus, and the global state has to include at least one component to originate the stimulus. An example occurs for the $Prune$ stimulus, for a global state $\{NH, F, NF\}$, where the an originating component of the $Prune$ (NC in this case) does not belong to the global state.
- Failure of minimum topology check: the necessary conditions to trigger the stimulus must be present in the global topology. Examples of failing the minimum topology check include $Join$ stimulus with global state $\{NH, NF\}$.
- Inconsistency: to maintain consistency of the transition rules in the reverse direction, we must check that every backward step has an equivalent forward step. We must check that there is no transition $x \rightarrow y$ for the given stimulus, such that $x \in GState$. Since if x remains in the preceding global state, the corresponding forward step would transform x into y and the system would exist in a state inconsistent with the initial global state (before the backward step) [17].

D. Applying The Method

The transition table is provided by the protocol designer [18], with a list of faults to be studied (e.g., selective loss of $\{Join, Prune\}$) and a set of initial states, in our case $\{NM, EU\}$. A definition of the correctness requirement should also be provided. The rest of the process is automated.

[15] 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.

[16] According to the transition completion concept, the proper analysis of behavior should start from externally triggered transitions. For example, the 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.

[17] An example of this inconsistency exists when the stimulus is $FPkt$ and $GState = \{F, NF, EU\}$, where $EU \rightarrow F$ is a post condition for $FPkt$.

[18] The traditional input/output transition table is sufficient for our method. The pre/post-condition transition table can be derived automatically therefrom.

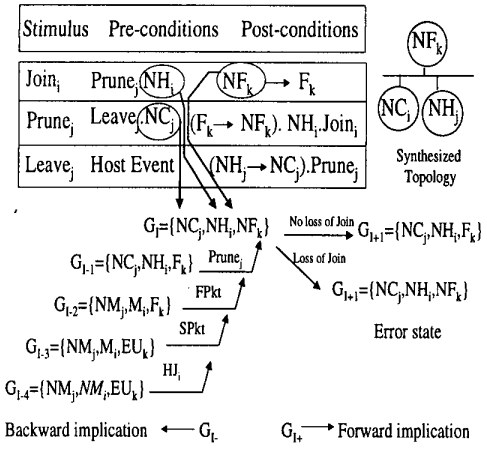

Fig. 3. $Join$ topology synthesis, forward/backward implication

Example. Figure 3 shows the phases of FOTG for $Join$ loss. Following are the steps taken for that example:

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 $NH_i \Rightarrow G_I = \{NH_i, NF_k\}$, 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 $NC_j \Rightarrow G_I = \{NH_i, NF_k, NC_j\}$, goto L (Ext)
5. $startState$ of post-condition is NH 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.

Summary of Results. We have studied single message loss scenarios for the $Join$, $Prune$, $Assert$, and $Graft$ messages. We also studied crash scenarios. For brevity, we partially discuss our results here. For a detailed analysis of the results see [3]. We have used the sequences of events generated automatically by the algorithm to analyze protocol errors and suggest fixes for those errors.

Join: A scenario similar to that presented in the illustrative example 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.

Limitations. Following are some limitations of FOTG.

- The topologies synthesized by the above FOTG study are only limited to a single-hop LAN with n routers. This means that the above FOTG analysis is necessary but not sufficient to verify robustness of the end-to-end behavior of the protocol; even if each LAN in the topology operates correctly, the inter-LAN interaction may introduce erroneous behaviors. Applying FOTG to multi-hop topologies is part of future research.
- The analysis for our case studies did not consider network delays. In order to study end-to-end protocols network delays must be considered in the model. In [4] we introduce the notion of *virtual* LAN to include end-to-end delay semantics.
- The topologies constructed by FOTG are inferred from the mechanisms specified by the transition table of the GFSM. The FOTG algorithm will not construct topologies resulting from non-specified mechanisms [19]. Extending FOTG to become 'error-oriented' test generation is part of our future work.
- The global states synthesized during the topology synthesis phase are not guaranteed to be reachable from an initial state. Hence the algorithm may be investigating unreachable states [20].

We believe that the strength of FOTG method is its ability to construct the necessary conditions for erroneous behavior by starting directly from the fault and avoiding exhaustive search. FOTG seems best fit to study protocol robustness in the presence of faults.

VI. Related Work

The related work includes protocol verification and VLSI design. Protocol verification addresses protocol properties, such as safety and liveness. In general, the two main approaches for protocol verification are theorem proving and reachability analysis [5]. In theorem proving, system properties are expressed in logic formulas, defining a set of axioms and constructing relations on these axioms. In contrast to reachability analysis, theorem proving can deal with infinite state spaces. Interactive theorem provers (e.g. VDM [6]) require hu-

[19] For example, if the $Assert$ mechanism that deals with duplicates was left out (due to a design error) the algorithm would not construct $\{F_i, F_j\}$ topology. Hence, FOTG is not guaranteed to detect duplicates in this case. So, FOTG may be used to evaluate behavior of specified mechanisms in the presence of network failures, but is not a general protocol verification tool.

[20] However, statistics collected in our case study [3] show that unreachable states are not the determining factor in the complexity of the backward search. Hence, other reduction techniques may be needed to increase the efficiency of the method.

man intervention, and hence are slow and error-prone. The number of axioms and relations grows with the complexity of the protocol. Moreover, these systems tend to abstract out network failures we are addressing in this study. Reachability analysis algorithms [7] attempt to generate and inspect all the protocol states that are reachable from given initial states. In general, such algorithm suffers from the 'state space explosion' problem. To circumvent this problem, state reduction and controlled partial search techniques [8] could be used. In our work we adopt approaches extending reachability analysis for multicast protocols. Our fault-independent test generation method borrows from controlled partial search and state reduction techniques.

VLSI design uses techniques for generation of test vector patterns. Test vector generation can be fault-independent or fault-oriented [9]. In the fault-oriented process, the main steps in generating a test vector are to excite the fault, and to propagate the resulting error to an observable output. Fault excitation and error propagation usually involve a search procedure with a backtracking strategy to undo contradiction in the assignment of line values. The line assignments performed may imply other line assignments. This process is referred to as *implication*. Forward implication is implying values of lines from the fault toward the output, while backward implication is implying values of lines from the fault toward the circuit input [21]. VLSI chip testing, however, is performed a given circuit, whereas protocol testing is performed for arbitrary and time varying topologies. Other related work includes verification of cache coherence protocols [10], which uses equivalence to reduce search complexity.

VII. Conclusions

In this study we have proposed the *STRESS* framework to integrate test generation into the protocol design process. Specifically, we targeted automatic test generation for robustness studies of multicast routing protocols. We have adopted a global FSM model to represent the multicast protocols on a LAN. In addition, we have used a fault model to represent packet loss and machine crashes. We have investigated two algorithms for test generation; namely, the fault-independent test generation (FITG) and the fault-oriented test generation (FOTG). Both algorithms were used to study a standard multicast routing protocol, PIM-DM, and were compared in terms of error coverage and algorithmic complexity. For FITG, equivalence reduction techniques were combined with forward search to reduce search complexity from exponential to polynomial. FITG does not provide topology synthesis. For FOTG, a mix of forward and backward search techniques allowed for automatic synthesis of the topology. We believe that FOTG is a better fit for robustness studies since it targets faults directly. The complexity for FOTG was quite manageable for our case study. Corrections to errors captured in the study were proposed with the aid of our method and integrated into the latest PIM-DM specification. More case studies are needed to show more general applicability of our methodology.

[21] Our approaches for protocol testing use some of the above principles; such as forward and backward implication.

References

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

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

[3] A. Helmy, S. Gupta, and D. Estrin. Systematic Testing of Multicast Routing Protocols: Analysis of Forward and Backward Search Techniques. *ACM-LANL-NCSTRL-cs.NI/0007005. http://xxx.lanl.gov/abs/cs.NI/0007005*, July 2000.

[4] A. Helmy, S. Gupta, D. Estrin, A. Cerpa, and Y. Yu. Systematic Performance Evaluation of Multipoint Protocols. *Proceedings of FORTE/PSTV, IFIP*, October 2000.

[5] 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.

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

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

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

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

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