STRESS Testing using Reduced Reachability Analysis: A Case Study for a Multicast Routing Protocol

Ahmed A-G Helmy, Deborah Estrin, Sandeep Gupta
Computer Science/Computer Engineering Department
University of Southern California
Los Angeles, CA 90089
e-mail:{ahelmy,estrin}@usc.edu, sandeep@boole.usc.edu

Abstract—Recent growth of the Internet and the introduction of new network services, have led to the increased complexity of network protocols and protocol interaction. In particular, the advent of IP-multicast has contributed to this added complexity. Multicast enables group communication in an efficient manner. Unlike traditional unicast protocols, a multicast group may involve multiple senders and receivers, and hence is inherently more complex. Multicast protocols support applications ranging from teleconferencing to network games.

In addition, increased heterogeneity of network components and technologies has introduced new failure modes that have not been considered traditionally in the design of multicast protocols; such as unicast routing anomalies and selective loss over LANs. The presence of these failures exacerbates the problem of designing and testing robust multicast protocols. To date, little effort has been exerted to formulate practical methods and tools that aid in the design and testing of these protocols.

In this paper we present an algorithm for automatic testing of multicast routing. The algorithm processes a finite state machine (FSM) model of the protocol and uses a forward search technique to generate the tests. We apply this algorithm to a simplified version of a multicast routing protocol; PIM-DM, to investigate its behavior in the presence of selective packet loss over LANs.

I. INTRODUCTION

A. Brief Overview of Multicast

Multicast protocols are the class of protocols that support group communication. A multicast group may involve multiple receivers and one or more senders. In this paper, we address multicast protocols for the Internet, based on the IP multicast model. These protocols include multicast routing protocols (e.g. DVMRP [1], MOSPF [2], PIM-DM [3], CBT [4], and PIM-SM [5]), multicast transport protocols (e.g. SRM [6], RTP, and RTCP [7]) and multiparty applications (e.g. WB [8], vat [9], vic [10], nte [11], and sdr [12]). This study focuses on multicast routing protocols, which deliver packets efficiently to group members by establishing distribution trees. Figure 1 shows a very simple example of a source $S$ sending to a group of receivers $R_i$.

Multicast distribution trees may be established by either broadcast-and-prune or explicit join protocols. In the former, such as DVMRP or PIM-DM, a multicast packet is broadcast to all leaf subnetworks. Subnetworks with no local members for the group send prune messages towards the source(s) of the packets to stop further broadcasts. Link state protocols, such as MOSPF, broadcast membership information to all
nodes. In contrast, in explicit join protocols, such as CBT or PIM-SM, routers send hop-by-hop join messages for the groups and sources for which they have local members. When received, these messages build routing state in routers, and cause further messages to be sent upstream until the distribution tree is established. Upon receiving a multicast packet, a router forwards the packet according to the routing state.

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

For most multicast protocols, when routers are connected via a multi-access network (or LAN)\(^1\), hop-by-hop messages are multicast on the LAN, and may experience selective loss; i.e., may be received by some nodes but not others. The likelihood of selective loss is increased by the fact that LANs often contain hubs, bridges, switches, and other network devices. Selective loss may affect protocol robustness.

Similarly, end-to-end multicast protocols and applications must deal with situations of selective loss. This differentiates these applications most clearly from their unicast counterparts, and raises interesting robustness questions.

Our case study illustrates why selective loss should be considered when evaluating protocol robustness. This lesson is likely to extend to the design of higher layer protocols that operate on top of multicast and can have similar selective loss.

II. Related Work

The related work falls mainly in the field of protocol verification. Most of the literature on multicast protocol design addresses architecture, specification, and comparisons between different protocols.

There is a large body of literature dealing with verification of communication protocols. Protocol verification typically addresses well-defined properties, such as safety, liveness, and responsiveness properties [13]. Safety properties include freedom from deadlocks, assertion violations, improper terminations and unspecified receptions. Liveness properties include detection of acceptance cycles and absence of non-progress cycles, while responsiveness properties include timeliness and fault tolerance. Most protocol verification systems aim to detect violations of (part of) these protocol properties.

In general, the two main approaches for protocol verification are theorem proving and reachability analysis (or model checking) [14], [15]. Theorem proving systems define a set of axioms and construct relations on these axioms. Desirable properties of the protocol are then proven mathematically. Theorem proving includes model-based formalisms (such as Z [16], and Vienna Development Method (VDM) [17]) and logic-based formalisms including first order logic (such as Nqthm [18]) and higher order logic (such as Prototype Verification System (PVS) [19]). An attempt to apply formal verification to TCP and T/TCP has been given in [20]. In general, however, the number of axioms and relations in theorem proving systems grows with the complexity of the protocol. We believe that these systems will be even more complex, and perhaps intractable, for multicast protocols. Moreover, these systems work with abstract specification of the protocol, and hence tend to abstract out some protocol mechanisms that may cause problems we are addressing in this study.

Reachability analysis algorithms [21], [22], on the other hand, try to generate and inspect all the protocol states that are reachable from given initial state(s). Such algorithms suffer from the 'state space explosion' problem, especially for complex protocols. To circumvent this problem, state reduction and controlled partial search techniques [23], [24] could be used. These techniques focus only on parts of the state space and may use probabilistic [25], random [26] or guided searches [27]. We note that some of the reduction techniques use 'equivalence' relations to prune away that part of the search proven to be irrelevant or redundant. Reduced reachability analysis has been used in the verification of cache coherence protocols [28], using a global FSM model. In our approach, we adopt the global FSM model and attempt to establish equivalence relations for the multicast routing protocol under study.

In [29] we presented a simulation-based STRESS framework, that utilizes heuristics and topological equivalence to establish the test scenarios. However, we did not address automatic generation of host scenarios. Our work in this paper complements and can be integrated with our previous work on STRESS. In [30] we have proposed a fault-oriented algorithm for the automatic generation of test scenarios. How-

\(^1\)We use the term LAN to designate a connected network with respect to IP-multicast. This includes shared media (such as Ethernet, or FDDI), hubs, switches, etc.
ever, the algorithm may investigate non-reachable states, and hence the complexity of the search may increase. On the other hand, the fault-oriented approach generates the topology as part of the output test, while the algorithm presented in this paper takes the topology as input.

III. Framework Overview

In general, the robustness of a protocol is its ability to respond correctly in the face of network failures and packet loss. This work presents an algorithm for studying and evaluating robustness of multicast routing protocols, with potential extension to multicast protocols in general. This section presents an overview of the test generation approaches. The model used to represent the protocol and the system is presented along with definitions of the terms used.

A. Test Generation

The core contribution of our work lies in the development of systematic test generation algorithms for protocol robustness. In general, there are two approaches for test generation (TG): random TG (RTG) and deterministic TG. RTG involves only the generation of random test patterns (see section III-B.3 for the definition of test patterns), and hence is simple. However, a large set of test patterns is needed to achieve a high measure of error coverage, and even then determining the test quality may be expensive. Also, the cost of running long test sequences may be high. RTG generally does not take into account the function or the structure of the protocol under test, and does not attempt to minimize the test length.

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

Deterministic TG can be: a) fault-independent, or b) fault-oriented.故障-oriented tests are generated for specified faults. Fault-oriented test generation starts from the fault (e.g., a lost message) and synthesizes the necessary conditions to drive the protocol into error. This algorithm uses a mix of forward and backward searches.

In contrast, 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. In this sense, it may be considered a variant of reachability analysis, with symbolic representation, and state and fault equivalence used to reduce the state space.

Throughout this document we adopt the fault-independent approach, extend it to fit our problem domain, and apply it to a variant of PIM-DM as an illustrative case study.

In the remainder of this section, we describe our system model and definition.

B. System Model and Definition

B.1 The system model

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

B.1.a Elements of the network. The network consists of links and nodes: routers and hosts. A link may be point-to-point or multi-access (e.g., LAN). In this document we assume bi-directional links with symmetric delays, while future work will address unidirectional and asymmetric links. A node runs a set of network protocols, such as unicast and multicast routing. We assume the existence of a MAC layer protocol to resolve media access and collision issues, but we do not model such protocol. A host runs end-to-end protocols or applications.

B.1.b Elements of the topology. In this document we consider only local topology [N-router LAN] modeled at the network level (i.e., connecting hubs, switches, bridges and other data-link-layer devices are abstracted out). The boundary of our topology is the multicast routing domain, which contains only a single multicast routing protocol. However, the topology may span multiple unicast routing domains or Autonomous Systems (ASs). Cascade of LANs or uniform topologies are mentioned in the future work section.

B.1.c The fault model. We distinguish between the terms error and fault. An error is a failure of the protocol as defined in the protocol design requirement and specification. For example, duplication in packet delivery is an error for multicast routing. A fault is a low level (e.g., physical layer) anomalous behavior, that may affect the behavior of the protocol under test, and include, for example packet loss or unicast route flapping, among others. Note that a fault may not necessarily be an error for the low level protocol.
The fault model may include:

- Loss of packets model packet loss on a link due to any queue congestion, overflow, link failures, or packet corruption in the interconnect devices, such as network interfaces, switches, hubs, etc. We assume that the packets are either delivered correctly, or are dropped; i.e. packet corruption is discovered using checksum or other error detection codes.
- Loss of state, such as multicast and/or unicast routing tables due to failure of the routing protocol, crashes, or insufficient memory resources.
- The delay model: Delays in the network may be due to transmission, propagation, or queuing delays. We assume that the processing delays are negligible with respect to the time granularity the analysis is addressing. Sometimes delay fault problems may be translated into event sequencing problems, as we will show by example later in this document.
- Unicast routing anomalies, such as route inconsistencies, oscillations or flapping.

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

Future work will consider other fault models, such as loss of state or unicast route flapping.

B.2 Test Sequence Definition

Given two sequences $T = \langle e_1, e_2, \ldots, e_n \rangle$ where $e_i$ is an event, and $T' = \langle e_1, e_2, \ldots, e_f, e_j, \ldots, e_n \rangle$, where $f$ 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 $q$. According to one of the following definitions $T'$ may be said to be a test sequence if:

1. $P(q, T) \neq P(q, T')$. This means that the behavior of the system in the presence of the fault is different than that without the fault. Note that this definition may include sequences that (including and excluding the fault) produce same correct final states, but with different transient behavior, or
2. Final $P(q, T) \neq P(q, T')$; i.e. the stable state (after the occurrence of the fault) is different for the two outputs. This definition ignores transient behavior, but may include sequences that (including and excluding the fault) produce different correct final states, or
3. Final $P(q, T')$ is incorrect 4; i.e. the stable state reached after the occurrence of the fault does not satisfy the correctness conditions, irrespective of $P(q, T)$. In case of a fault-free sequence, where $T = T'$, the error is attributed to a protocol design error. Whereas when $T \neq T'$, and final $P(q, T)$ is correct, the error is manifested by the fault.

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

B.3 Test Input Pattern

A test input pattern is defined by a list of (host) events $Ev$, a topology $T$, and a fault model $F$. We define a test input pattern as a 3-tuple $< Ev, T, F >$:

- Events: $Ev = < ev_1, ev_2, \ldots, ev_n >$ is a list of host events (host scenarios, or call patterns). Each event $ev_j$ consists of $< action, time >$, where action: is the host (or node) event input; for example, join, leave, send packet, etc.
- Topology: $T = < N, L >$ is the routed topology of set of nodes $N$ and links $L$. $N = < n_1, n_2, \ldots, n_k >$, is the list of nodes each running a set of protocols. A protocol may be modeled by $< timers, messages, stateVars, mechanisms >$.
- Faults: $F$ is the fault model used to inject the fault into the test. According to our single-message loss model, for example, a fault may denote the 'loss of the second message traversing link $l_i$ of type prune'. Knowing the location and the triggering action of the fault is important in analyzing the protocol behavior.

As a case study, we apply our automatic test generation algorithms to a version of the Protocol Independent Multicast-Dense Mode, or PIM-DM.

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.

B.3.a PIM Protocol Errors. In this study we target protocol design and specification errors. We are interested mainly in erroneous stable (i.e. non-transient) states. We assume that these errors are provided by the protocol designer or the protocol specification.

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

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

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

These are the correctness conditions for stable states; i.e. not during transients, and are defined in terms of protocol states (as opposed to end point behavior). They are used in the fault-independent and fault-oriented test generation, where the protocol model does not capture end point traces. We also use these conditions for topological equivalence in the heuristic test generation.

B.4 The Protocol Model

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

B.4.a FSM model. A deterministic finite state machine modeling the behavior of a router R_i is represented by the machine \( M_i = (Q, \Sigma_i, \delta_i) \), where
- \( Q \) is a finite set of state symbols
- \( \Sigma_i \) is the set of operations causing state transitions; and
- \( \delta_i \) is the state transition function \( Q \times \Sigma_i \rightarrow Q \).

B.4.b Global FSM model. With respect to a particular LAN, the global state is defined as the composition of individual router states w.r.t. to that LAN. The behavior of a LAN with \( n \) routers may be described by the global FSM \( M_G = (Q_G, \Sigma_G, \delta_G) \) where
- \( Q_G: Q_1 \times Q_2 \times \cdots \times Q_n \) is the global state space,
- \( \Sigma_G = \bigcup_{i=1}^n \Sigma_i \) is the set of operations causing the transitions; and

Some esoteric scenarios of route flapping may lead to multicast loops, in spite of RPF checks. Currently, our study does not address this issue, as it does not pertain to a localized behavior.
\[ \delta_{G} : \text{is the global state transition function } Q_{G} \times \Sigma_{G} \rightarrow Q_{G} \]

IV. Fault-independent Test Generation

In this section, we investigate a fault-independent approach for automatic test generation. We use a variant of reachability analysis to inspect a reduced subset of the state space of the system for errors. We develop equivalence relations and symbolic representation techniques to reduce the complexity of the used algorithm from exponential to linear in the number of routers on the LAN. To examine the robustness of the protocol against single message loss, we incorporate selective loss scenarios in the inspected space.

To illustrate the procedures of this method we apply it to a version of PIM-DM.

A. Formalism

A formalism is used to represent the protocol as a finite state machine (FSM). A multi-access LAN is used as the target system and is represented by a global FSM.

We use the FSM model explained earlier in section III-B.4 to model the protocol.

A.1 FSM model

\[ M_{i} = (Q, \Sigma_{i}, \delta_{i}) \]

A.1.a System States (Q). We define the states w.r.t. a specific LAN to which the router \( R_{i} \) is attached. A router is represented by its state as a forwarder or a receiver of packets to or from the LAN.

The possible states are described in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F )</td>
<td>Router ( i ) forwarder for the LAN</td>
</tr>
<tr>
<td>( NH )</td>
<td>Router ( i ) has the LAN as its next-hop</td>
</tr>
<tr>
<td>( NC )</td>
<td>Router ( i ) has a negative-cache entry</td>
</tr>
<tr>
<td>( E )</td>
<td>Router ( i ) does not have an entry; i.e. is empty</td>
</tr>
</tbody>
</table>

We also differentiate whether a router is upstream or downstream. For example,

\[
Q = \begin{cases} 
\{ E, F \} & \text{if the router is upstream,} \\
\{ E, NH, NC \} & \text{if the router is downstream.} 
\end{cases}
\]

where the event with parenthesis indicates the transmission of a message and the event without indicates reception. \( G = \text{Graft}, J = \text{Join}, Pk = \text{Packet}, Pr = \text{Prune}, L = \text{Leave} \).

Figure 2 gives a finite state machine for one node implementing (a simplified version of) PIM-DM. When an upstream router receives data packets (from the next-hop neighbor), a Graft or Join (from downstream), it creates a forwarding state \( \{ F \} \). If the upstream router receives a Prune from downstream, it removes the forwarding entry. For a downstream router, data packets trigger the creation of the state as follows: (a) if there exists downstream members, a receiving state (i.e. \( \{ NH \} \)) is created and a Graft is triggered upstream, else (b) a negative cache state \( \{ NC \} \) is created and a Prune is triggered upstream. Receiving a Graft or Join creates a receiving state and triggering a Graft upstream, while receiving a Prune or Leave creates a negative cache state and triggers a Prune upstream. Asserts are not shown for simplicity.

A.2 Global FSM model

An example global state for a topology of 5 routers connected to a LAN, with router 1 as a forwarder, router 2 expecting packets from the LAN, router 3 and 4 have negative caches, and router 5 is empty,
is given by \( \{ F_1, NH_2, NC_3, NC_4, E_5 \} \). Conveniently, we may omit the empty state \( E_5 \).

We have two possible state symbols for upstream routers and three for downstream routers, hence the total number of possible global states for \( n \)-router LAN is \( |Q_j| = 2^{[\text{upstream}]} \times 3^{[\text{downstream}]} \), where \( [\text{upstream}] + [\text{downstream}] = n \).

B. Fault-independent Test Generation Algorithm

Our approach employs a variant of reachability analysis to investigate the global FSM model developed for the system in section III-B.4. We start from (a subset of) the correct states, and investigate the system transitions according to various operators in addition to selective loss. The output (if any) is a set of event sequences that drive the system into erroneous states. The approach includes the following steps:

1. Establish the correct states for the given topology according to the correctness conditions. We use equivalence techniques to reduce the number of correct states to be inspected.
2. Start from a correct state, and search for a sequence of transitions (possibly including a selective loss scenario) that lead to an erroneous state (i.e., those non-transient states in which correctness conditions are violated). The transitions obtained are those defined for the FSM, and not necessarily host events (such as Join or Leave). The number of loss scenarios investigated is reduced based on equivalence relations.
3. Establish external host events leading to the transitions obtained by the previous step. Call this sequence of events \( S_{err} \). The sequence of host events in this case may not be unique. A prune, for example, may be triggered due to various events (a joined host leaving or receiving packets with no downstream members).
4. Establish external host events leading to the correct state. Again, such sequence of events may not be unique. Call this sequence \( S_{corr} \).
5. The input test pattern of events is the concatenation of \( S_{corr} \) and \( S_{err} \); i.e., \( S_{corr} \cdot S_{err} \).

C. Obtaining Correct States

Theoretically, all states that satisfy the correctness conditions may be considered correct states. However, not all of these states are normally reachable. For example, all downstream members build state whenever they get a data packet (whether it is a receiving state \( NH \) or negative cache \( NC \)), hence, for a topology of one upstream router '1' and two downstream routers '2' and '3', the global state \( \{ F_1, NH_2, E_3 \} \), although theoretically correct, is practically unreachable. We focus on the practically reachable correct states in this section, and develop the following algorithm to obtain those states:

ReachableCorrectStates(upstrm,dnstrm)
add EmptySet to setOfStates;
\forall j \in \text{dnstrm}
add NC\(_j\) to State;
add State to setOfStates;
\forall i \in \text{upstrm} \{
for mask=1 to \(2^{[\text{dnstrm}]}\) {
add \( F_i \) to State;
position = 0;
\forall j \in \text{dnstrm} \{
if (1 << position) & mask == 1
add NH\(_j\) to State;
else
add NC\(_j\) to State;
position = position+1;
}
add State to setOfStates;
}
}

The set of correct states (or C.S.) is a subset of the state space (i.e., \( C.S. \subseteq Q_j \)). The number of reachable correct states generated by the algorithm is \(|C.S.| = [\text{upstream}] \cdot (2^{[\text{downstream}]} - 1) + 2\).

Reachability Analysis and Reduction Techniques:
- Exhaustive search: Attempts to generate and analyze all system states that are reachable from an initial system state.
In a system of \( n \) routers, \(|Q| = m \) states for a router, the number of reachable states in the system is bounded by \((m)^n\). To investigate all the transitions, with \(|\Sigma_j| = l \) possible transitions, we obtain \( l \cdot (m)^n \) state visits to complete the process.

\(^7\)under specific crash scenarios, this state may be reached. But with the single-fault model we have adopted, this state cannot be reached, as we assume, without loss of generality, that the fault will occur after the correct states (one of which being the initial state) is reached.
• Symbolic representation
An alternative representation of the system may be obtained through symbolic representation, where \( r \) routers in state \( q \) are represented by \( q^r \). The global state for a system of \( n \) routers is represented by \( S = (q_1^r, q_2^r, \ldots, q_m^r) \), where \( m = |Q| \), \( \Sigma r_i = n \) and \( r_i \in \{0,1,1+,\} \) (\( 1^+ \) is 1 or more, and \( ^* \) is 0,1, or more). For our case study \( q_i \in \{F, NH, NC, E\} \).

To satisfy correctness conditions 1, 2, and 4, given in section III-B.3.b, the correct stable global states are those containing no forwarding and no routers expecting packets, or those containing one forwarder and one or more routers expecting packets from the link; symbolically this is given by:

\[
S = (F^0, NH^0, NC^*, E^*) \quad \text{and} \quad S = (F^1, NH^1+, NC^*, E^*).
\]

• Counting equivalence: Two system states \((q_1, q_2, \ldots, q_n)\) and \((p_1, p_2, \ldots, p_n)\) are strictly equivalent iff \( q_i = p_i, q_i, p_i \in Q, \forall 1 \leq i \leq n \). However, the behavior of all routers is given by a common deterministic FSM, hence all \( n! \) permutations of a \((q_1, q_2, \ldots, q_n)\) are equivalent because the order of the tuple is not important.

A state for a system with \( n \) routers may be represented as \( \prod_{i=1}^{n} q_i^{k_i} \), where \( k_i \) is the number of routers in state \( q_i \) and \( \Sigma_{i=1}^{n} k_i = n \). Counting Equivalence: Two system states \( \prod_{i=1}^{n} q_i^{k_i} \) and \( \prod_{i=1}^{n} q_i^{l_i} \) are equivalent if \( k_i = l_i \forall i \).

In other words, two system states are equivalent if the number of routers in a specific state in one system is equal to the number of routers in the same state in the other system, for all router states.

In our analysis we do not attempt to investigate the whole state space, rather, we start from the correct states (under relaxed practical assumptions) and analyze the successor (stable) states. However, even with this approach (i.e. the reachable correct states algorithm) the number of correct states grows exponentially with the number of downstream routers. In addition, we need to investigate various loss scenarios, and that is also exponential in the number of routers.

We use the counting equivalence to reduce the number of correct states explored, and the number of investigated loss scenarios.

1. Exploring correct states: From the counting equivalence relationship, and the symmetry given in the LAN topology, we can reduce the complexity of the algorithm. For example, the state \( \{F_1, NH_2, NC_3\} \) is equivalent to \( \{F_1, NC_2, NH_3\} \), since both of these states corresponds to the symbolic representation \( F^1 \cdot NH^1 \cdot NC^1 \). Using this equivalence relation, we modify the correct states algorithm as follows:

\[
\text{ExploredCorrectStates}(\text{upstr}, \text{dnstr}) \{
\begin{align*}
\text{add EmptySet to setOfStates;}
\forall j \in \text{dnstr} \quad & \text{add } NC_j \text{ to State;}
\text{add State to setOfStates;}
\forall i \in \text{upstr} \{
& \text{for mask}=1 \text{ to } |\text{dnstr}| + 1 \\
& \quad \text{add } F_i \text{ to State;}
& \quad \text{position} = 0;
& \quad \forall j \in \text{dnstr} \{
& \quad \quad \text{if } (\text{position} < \text{mask})
& \quad \quad \quad \text{add } NH_j \text{ to State;}
& \quad \quad \text{else}
& \quad \quad \quad \text{add } NC_j \text{ to State;}
& \quad \quad \quad \text{position} = \text{position} + 1;
& \quad \text{add State to setOfStates;}
\}
\}
\}
\]

An example output of the above algorithm, for an upstream router \( R_1 \) and three downstream routers \( R_2, R_3 \) and \( R_4 \) would be:

\[
\{\}, \{NC_2, NC_3, NC_4\}, \{F_1, NH_2, NC_3, NC_4\}, \{F_1, NH_2, NH_3, NC_4\}, \{F_1, NH_2, NH_3, NH_4\}.
\]

We note that the new algorithm has a complexity of \( O(|\text{upstreamNodes}| \cdot |\text{downstreamNodes}|) \) as opposed to \( O(|\text{upstreamNodes}| \cdot 2^{\text{downstreamNodes}}) \) for the previous algorithm. We can further reduce this complexity by considering all upstream routers equivalent, hence we can remove the ‘\( \forall i \in \text{upstreamNodes} \)’ loop, which gives \( O(|\text{downstreamNodes}|) \) complexity.

2. Exploring selective loss scenarios: In general, similar equivalence relation may be applied to the selective loss patterns explored. For example, a prune sent by a downstream router \( R_i \) may be lost by either downstream routers \( R_j \) and \( R_k \). If both these routers have the same state (e.g. \( NH_j \in \text{GlobState} \) and \( NH_k \in \text{GlobState} \)), then the scenarios \( \{R_{\text{lostPrune}}, R_j, R_{\text{revPrune}}\} \) and \( \{R_{\text{lostPrune}}, R_j, R_{\text{revPrune}}\} \) are equivalent.\(^9\)

\(^8\) For convenience, we omit the \( E \) state.

\(^9\) By looking closely at the prune mechanism, we find that an interesting scenario (or distinguishing scenario) is one where \( R_{\text{lostPrune}} \cdot \prod_i R_{\text{lostPrune}} \) is s.t. \( NH_i \in \text{GlobState}. \)
D. Obtaining Stable States

To establish the 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. Then we present the protocol mechanisms and the stable state checking algorithm.

D.1 Classification of Transitions

We identify two types of transitions: externally triggered (ET) and internally triggered (IT) transitions. The first type is stimulated by actions external to the system (such as host-join or host-leave), whereas the second type is stimulated by actions internal to the system (such as prune and assert).

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

The global state is checked for correctness only at the end of an ET transition and after completing all dependent ITs.

Following is a table of host events, their dependent ETs and their dependent ITs:

<table>
<thead>
<tr>
<th>Host Events</th>
<th>ETs</th>
<th>ITs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Forward</td>
<td>Assert, Prune</td>
</tr>
<tr>
<td>Host.Join</td>
<td>Graft</td>
<td>Prune</td>
</tr>
<tr>
<td>Leave</td>
<td>Prune</td>
<td>Join</td>
</tr>
</tbody>
</table>

D.2 FSM Reduction

Consider the transition diagram and topology in figure 3. When the upstream router (1) receives data packets, it becomes the forwarmer $\{F_1\}$. Depending on downstream membership status, router (2) either (a) creates a receiving state $NH_2$, moving the global state to $\{F_1,NH_2\}$, or (b) creates a negative cache state $NC_2$ (i.e., global state $\{F_1,NC_2\}$), then sends a Prune upstream, resulting in router (1) removing its forwarding state, and moving the global state to $\{NC_2\}$.

If a downstream member joins, router (2) creates a receiving state $\{NH_2\}$ and send a Graft upstream, causing router (1) to create a forwarding state, and the global state becomes $\{F_1,NH_2\}$.

We note that Grafts are sent reliably; hence the state $\{NH_2\}$ is a transient state, and the transition from state $\{NC_2\}$ to $\{F_1,NH_2\}$ is considered reliable.

Similarly, data-triggered actions (such as prunes on point-to-point links, and creation of downstream state; whether receiving or negative cache) may be considered reliable. Subsequently, the states $\{F_1\}$ and $\{F_1,NC_2\}$ are also considered transient states.

As was alluded to earlier, transient states are not checked for correctness, and hence may be removed from the diagram. The resulting transition diagram is shown in figure 4.

D.3 Transition Completion

From the previous section we showed that transient states can be removed from the inspected state space, based upon the reliability by which the transition out of these states is accomplished. It seems as though the error can be detected by inspecting the non-transient state space. To investigate the efficacy of such approach we analyze a topology with some un-reliable transitions; such as that given in figure 5.

The reader is encouraged to view the figure in details. However, we only concentrate on the transitions affected by loss. One such transition is shown in figure 6.

It is apparent from the figure that the global state
\{NH_2, NC_3\} can be considered a transient state (that may transit to a correct state \{F_1, NH_2, NC_3\} in some cases, and a stable error state in others, and hence cannot be simply removed from the state machine. At the same time, an error is not encountered by the mere reachability of that state; i.e. the state should not be always checked for correctness. Which leads us to introduce the notion of transition completion explained next.

To check for the global system correctness, all stimulated internal transitions should be completed, to bring the system into a stable (i.e. non-transient) state. Intermediate (transient) states should not be checked for correctness (since in most cases they vi-
ulate 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. In general, however, we may identify a complete transition sequence, as the sequence of (all) transitions triggered due to a single external stimulus (e.g. host-join or host-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. This transition completion concept suggests that starting from a correct state, only one complete transition sequence needs to be explored. Hence, if all the correct states (or a subset equivalent thereto) are investigated, all erroneous states will be discovered.

A phase contains all transitions that are considered complete. For PIM-DM the following sequences of transitions are considered complete (for the purposes of this study):

- sending prunes then sending joins [if any] (or P.J. for short)
- sending packets, prune [if any], join [if any], asserts [if any] (S.P.J.A).

A P.J. complete phase does not mean that a join cannot be lost, rather, any loss pattern may be applied to the join message, but the processing of the join (after applying the loss pattern) should be completed before the state is checked for correctness.

E. Protocol mechanisms

Following are the mechanisms representing the PIM-DM protocol from the perspective of a router \( R_i \) connected to the link. For brevity, we simplify some of the protocol mechanisms, such as assuming Grafts to be reliable and not considering timers.

1. Pruning:
   
   procedure SendPrune\(_{i \rightarrow dst}\)
   
   if \( F_{dst} \in \text{GlobState} \)
   
   remove \( F_{dst} \) from \( \text{GlobState} \);
   
   join = 0;
   
   \( \forall j \in \text{downstream} & j \neq i \)
   
   /* apply selective loss */
   
   if \( NH_j \in \text{GlobState} \)
   
   join ++;
   
   if (join)
   
   SendJoin();
   
   CheckState(); /* end of a phase */

2. Joining (Prune-overriding):
   
   procedure SendJoin\(_{i \rightarrow dst}\)
   
   /* apply message loss */
   
   if \( F_{dst} \notin \text{GlobState} \)
   
   add \( F_{dst} \) to \( \text{GlobState} \);
   
   CheckState(); /* end of a phase */

3. Grafting:
   
   procedure Send Graft
   
   if \( F_{dst} \notin \text{GlobState} \)
   
   add \( F_{dst} \) to \( \text{GlobState} \);
   
   CheckState(); /* end of a phase */

4. Asserting:
   
   procedure Send Assert
   
   \( \forall j \in \text{upstream} \)
   
   if \( j \neq \text{max} /* max address wins assert */
   
   remove \( F_j \) from \( \text{GlobState} \);
   
   assert = 0;
   
   \( \forall k \in \text{upstream} \)
   
   if \( F_k \notin \text{GlobState} \)
   
   add \( F_k \) to \( \text{GlobState} \);
   
   assert ++;
   
   if (assert)
   
   SendAssert();
   
   prune = 0;
   
   \( \forall j \in \text{downstream} \)
   
   if \( NC_j \in \text{GlobState} \)
   
   prune ++;
   
   else if \( NH_j \notin \text{GlobState} \)
   
   add \( NH_j \) to \( \text{GlobState} \);
   
   if (prune)
   
   SendPrune();
   
   CheckState(); /* end of a phase */

6. CheckState(\( \text{GlobState} \))
The externally triggered events from the above list are the forwarding and the loss events. The ForwardPacket$_1$ maps into a sendPacket$_1$ host event. Hence, $S_{corr} = \{\text{sendPacket}_1, R_1\text{lostJoin}\}$.

To get to the correct state we have $S_{corr} = \{\text{HostJoin}_2, \text{Leave}_3, \text{sendPacket}_1\}$.

The test sequence is:

$\{\text{HostJoin}_2, \text{Leave}_3, \text{sendPacket}_1, R_1\text{lostJoin}\}$

V. SUMMARY AND FUTURE WORK

In this paper we have presented an algorithm for automatic test generation that targets protocol robustness. We have used a LAN model, to study protocol behavior in the presence of selective loss. Our method was applied to a simplified version of PIM-DM. For our case study, we have achieved state space reduction from exponential to linear complexity in the number of routers connected to the LAN. We have presented an example where errors were discovered in the protocol behavior in the presence of message loss, and the events leading to the error were generated. The methodology may be applied to other faults (e.g., unicast routing anomalies or router crashes) as well as other protocols that may be represented using the GFSM model used in this paper.

REFERENCES


