SYSTEMATIC TEST SYNTHESIS FOR MULTIPOINT PROTOCOL DESIGN

by

Ahmed Abdel-Ghaffar Helmy

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
( Computer Science )

August 1999

Copyright 1999 Ahmed Abdel-Ghaffar Helmy
Dedication

To my parents ‘I owe you every success’.
To my father Abdel-Ghaffar Helmy, you were a father, a brother and a dearest friend,
and to my mother Hadia Ramadan, you supported me all my life to discover my
potentials. I will always be indebted and grateful to you.

To my brother Tarek Helmy, you are always a big brother and a friend.
To my wife-to-be and best friend for life Hania Abou Al-Shamat, you give my life balance
and meaning, and you complete my being.

A special note to my aunt Hiba Ramadan Attar, you are a model for altruism and
patience, and you make everything in this life seem so simple. You have a heart of gold.

With all the love and respect to my family.
Acknowledgments

I am grateful to Deborah Estrin, my advisor, mentor and friend. Deborah’s support, encouragement and guidance lead the way for me to realize my ideas through our research. Deborah’s understanding in technical and personal matters was a motive for me to continue through tough times.

Special thanks go to Sandeep Gupta, my co-advisor for the STRESS project. His insightful comments and deep questions made us re-think lots of our initial research ideas and improve upon them.

Thanks for my committee members, Deborah Estrin, Sandeep Gupta and Ramesh Govindan, for their useful discussions and comments on the dissertation and for serving on my qualification and defense exams.

Thanks also go to Cengiz Aläettinoglu and Maja Mataric for serving on my qualification committee in addition to the above members.

I would like to thank all my colleagues in the PIM, VINT and STRESS projects. Their interaction enriched my experience and inspired me. Especially, I would like to thank Reza Rejaie for being such a dear friend in good and bad times.

I thank my colleagues at ISI, Cisco, and Silicon Graphics Inc, and all those who participated in ideas or discussion useful to this work.
## Contents

Dedication ii  
Acknowledgments iii  
List Of Figures viii  
Abstract x  

1 Introduction 1  

2 Related Work 3  
  2.1 Multicast Protocols ........................................ 4  
  2.1.1 Multicast Routing ...................................... 4  
  2.1.2 End-to-end Multipoint Protocols ......................... 6  
  2.2 Protocol Verification ..................................... 6  
  2.2.1 Theorem Proving ....................................... 7  
  2.2.2 Reachability analysis ................................ 8  
  2.3 Distributed Algorithms ................................... 11  
  2.4 Conformance Testing ...................................... 13  
  2.5 VLSI Chip Testing ....................................... 14  

3 Framework Overview 17  
  3.1 Framework Overview ....................................... 17  
  3.1.1 Test Generation ...................................... 18  
  3.1.2 Detailed Simulation .................................. 20  
  3.1.3 Implementation Interface ............................... 20  
  3.2 System Model and Definition ................................ 20  
  3.2.1 The system model ..................................... 20  
  3.2.2 Test Sequence Definition .............................. 22  
  3.2.3 Test Input Pattern ................................... 22  
  3.2.4 Test requirement ..................................... 23  
  3.2.5 Brief description of PIM-DM ....................... 23
### 4 Heuristic Test Generation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Method Overview</td>
<td>26</td>
</tr>
<tr>
<td>4.1.1 Scenario Generation</td>
<td>27</td>
</tr>
<tr>
<td>4.1.2 Simulation and Tracing</td>
<td>29</td>
</tr>
<tr>
<td>4.1.3 Output Analysis</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Case Study</td>
<td>31</td>
</tr>
<tr>
<td>4.2.1 Topology Equivalence</td>
<td>32</td>
</tr>
<tr>
<td>4.2.2 Test suites</td>
<td>36</td>
</tr>
<tr>
<td>4.3 Applying the Method</td>
<td>37</td>
</tr>
<tr>
<td>4.3.1 Obtaining faulty scenarios</td>
<td>37</td>
</tr>
<tr>
<td>4.3.2 Example for PIM-DM</td>
<td>38</td>
</tr>
</tbody>
</table>

### 5 Search-based Approaches

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 The Protocol Model</td>
<td>40</td>
</tr>
<tr>
<td>5.2 PIM-DM Model</td>
<td>41</td>
</tr>
<tr>
<td>5.2.1 FSM model $M_i = (S_i, T_i, \delta_i)$</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2 Global FSM model</td>
<td>43</td>
</tr>
<tr>
<td>5.3 Defining stable states</td>
<td>43</td>
</tr>
<tr>
<td>5.3.1 Classification of Transitions</td>
<td>43</td>
</tr>
<tr>
<td>5.3.2 Transition Completion</td>
<td>44</td>
</tr>
<tr>
<td>5.4 Problem Complexity</td>
<td>44</td>
</tr>
<tr>
<td>5.4.1 Complexity of exhaustive state space search</td>
<td>44</td>
</tr>
<tr>
<td>5.4.2 State reduction through equivalence</td>
<td>45</td>
</tr>
<tr>
<td>5.4.3 Representation of error and correct states</td>
<td>46</td>
</tr>
</tbody>
</table>

### 6 Fault-independent Test Generation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Exhaustive Search</td>
<td>49</td>
</tr>
<tr>
<td>6.2 Reduction Using Equivalences</td>
<td>50</td>
</tr>
<tr>
<td>6.3 Applying The Method</td>
<td>50</td>
</tr>
<tr>
<td>6.3.1 Method input</td>
<td>51</td>
</tr>
<tr>
<td>6.3.2 Complexity of forward search for PIM-DM</td>
<td>51</td>
</tr>
<tr>
<td>6.3.3 Summary of behavioral errors for PIM-DM</td>
<td>52</td>
</tr>
<tr>
<td>6.4 Challenges and Limitations</td>
<td>54</td>
</tr>
</tbody>
</table>

### 7 Fault-oriented Test Generation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Method Overview</td>
<td>56</td>
</tr>
<tr>
<td>7.1.1 Transition Table</td>
<td>57</td>
</tr>
<tr>
<td>7.2 FOTG details</td>
<td>60</td>
</tr>
<tr>
<td>7.2.1 Synthesizing the Global State</td>
<td>60</td>
</tr>
<tr>
<td>7.2.2 Forward Implication</td>
<td>61</td>
</tr>
<tr>
<td>7.2.3 Backward Implication</td>
<td>62</td>
</tr>
<tr>
<td>7.3 Applying The Method</td>
<td>63</td>
</tr>
<tr>
<td>7.3.1 Method input</td>
<td>63</td>
</tr>
<tr>
<td>7.3.2 Illustrative example</td>
<td>63</td>
</tr>
<tr>
<td>7.3.3 Summary of Results</td>
<td>65</td>
</tr>
<tr>
<td>7.4 Challenges and Limitations</td>
<td>67</td>
</tr>
</tbody>
</table>
Appendix C
Forward Search Algorithms .............................................. 116
C.1 Exhaustive Search .................................................... 116
C.2 Reduction Using Equivalence ........................................ 117
C.3 Complexity analysis of forward search for PIM-DM .......... 119

Appendix D
FOTG Algorithms ......................................................... 122
D.1 Pre-Conditions ....................................................... 122
D.2 Dependency Table .................................................... 122
D.3 Topology Synthesis ................................................... 123
D.4 Backward Search ..................................................... 123
D.5 Experimental statistics for PIM-DM .............................. 126
D.6 Results .............................................................. 127
D.7 Interleaving events and Sequencing .............................. 130

Appendix E
End-to-End Performance Evaluation ................................. 132
E.1 Conditions and Inequalities for Overhead Analysis ........ 132
   E.1.1 Worst-case Overhead Analysis ................................ 133
   E.1.2 Best-case Analysis ............................................ 134
E.2 Mathematical Model for Solving the System of Inequalities .. 135
E.3 Multiple request rounds ............................................. 136
## List Of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Establishing multicast delivery tree</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Generic BIST scheme</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>The STRESS framework</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Test pattern dimensions</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>The block diagram of the simulation method</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>The scenario filter</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>The equivalent topologies</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>The topology used for the case study</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>The prune – override loss scenario for PIM-DM</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>The percentage of the correct and error states</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>Simulation statistics for forward algorithms. Expanded States is the number of visited states and Forwards is the number of forward advances of the state machine.</td>
<td>52</td>
</tr>
<tr>
<td>6.2</td>
<td>Simulation statistics for forward algorithms. Transitions is the number of transient states visited, and Errors is the number of stable state errors detected.</td>
<td>53</td>
</tr>
<tr>
<td>7.1</td>
<td>Join topology synthesis, forward/backward implication</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>Craft event sequencing</td>
<td>66</td>
</tr>
<tr>
<td>7.3</td>
<td>Crash leading to join latency</td>
<td>67</td>
</tr>
<tr>
<td>7.4</td>
<td>Crash leading to black holes</td>
<td>68</td>
</tr>
<tr>
<td>8.1</td>
<td>The virtual LAN and the delay matrix</td>
<td>73</td>
</tr>
<tr>
<td>8.2</td>
<td>Time lines showing possible event sequencing: (a) and (b) sequences do not lead to suppression, while (c) leads to timer suppression.</td>
<td>82</td>
</tr>
<tr>
<td>8.3</td>
<td>The virtual LAN with 3 potential responders</td>
<td>86</td>
</tr>
<tr>
<td>8.4</td>
<td>The virtual LAN, showing pair-wise delays</td>
<td>87</td>
</tr>
<tr>
<td>A.1</td>
<td>How senders rendezvous with receivers</td>
<td>100</td>
</tr>
<tr>
<td>A.2</td>
<td>The topology used for the case study</td>
<td>102</td>
</tr>
<tr>
<td>A.3</td>
<td>Simple packet trace graph showing packet loss and duplication</td>
<td>103</td>
</tr>
<tr>
<td>A.4</td>
<td>The Assert scenario under study</td>
<td>107</td>
</tr>
<tr>
<td>A.5</td>
<td>Transition diagram for joins and asserts</td>
<td>108</td>
</tr>
</tbody>
</table>
A.6 The Join/Prune scenario under study .................................. 109
A.7 The register scenarios under study .................................... 110

C.1 Simulation statistics for forward algorithms. ExpandedStates is the number
of visited states. ................................................................. 118
C.2 Simulation statistics for forward algorithms. Forwards is the number of
calls to forward(). ............................................................. 119
C.3 Simulation statistics for forward algorithms. Transitions is the number of
transient states visited ...................................................... 120
C.4 Simulation statistics for forward algorithms. The number of stable error
states reached. ................................................................. 121
C.5 Reduction ratio from exhaustive to the reduced algorithm ............ 121

D.1 Simulation statistics for backward algorithms .......................... 127
D.2 Complexity of the FOTG algorithm for error states ................. 128
D.3 Percentage of reachable/unreachable error states using FOTG ....... 129
D.4 A topology having a \{F_i, F_j, \ldots, F_k\} LAN ....................... 130
D.5 Graft event sequencing .................................................... 131

E.1 The time-line for transition ordering ................................... 133
E.2 Forward search for multiple simultaneous events .................... 137
E.3 Backward search for multiple simultaneous events ................... 138
Abstract

The recent growth and increased heterogeneity of the Internet has increased the complexity of network protocol design and testing. In addition, the advent of multipoint (multicast-based) applications has introduced new challenges that are qualitatively different in nature than the traditional point-to-point protocols. Multipoint applications typically involve a group of participants simultaneously, and hence are inherently more complex. As more multipoint protocols are coming to life, the need for systematic and automated methods to study and evaluate such protocols is becoming more apparent. Such methods aim to expedite the protocol development cycle and improve protocol robustness and performance.

Related work on automatic protocol verification has usually targeted protocol correctness under idealized conditions. This study targets protocol robustness and performance in the presence of lower level network failures. In addition, previous work does not address multicast protocols or topology synthesis, both of which are addressed in this study.

In this dissertation a new methodology for developing systematic and automatic test generation algorithms for multipoint protocols is proposed. 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 research is to circumvent the state space explosion problem utilizing knowledge of network and fault modeling, and multipoint protocols. Several approaches are investigated in this dissertation, including approaches based on heuristic, forward and backward search techniques. Focus is given on two search algorithms based on 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, automates topology synthesis by utilizing backward
search techniques. This algorithm uses backtracking to generate event sequences instead of searching forward from initial states.

Using these algorithms, studies are conducted for correctness of multicast routing protocols over LANs. These algorithms are further extended to study end-to-end multipoint protocols, by incorporating delay semantics and performance criteria. The notion of a virtual LAN is introduced to represent delays of the underlying multicast distribution tree. As a case study, the method is used to generate worst and best case scenarios for the timer suppression mechanism employed in several Internet multipoint protocols.
Chapter 1

Introduction

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

Protocol errors may be very costly to repair if discovered after deployment. Hence, endeavors should be made to capture protocol flaws early in the design cycle before deployment.

Many researchers have developed protocol verification methods, to ensure that certain properties of a protocol hold; properties like freedom from deadlocks or unspecified receptions. Much of this work, however, was based on assumptions about the network conditions, that may not always hold in today’s Internet, and hence may become invalid. Other approaches, such as reachability analysis, attempt to check the protocol state space, and generally suffer from the ‘state explosion’ problem. This problem is exacerbated with the increased complexity of the protocol. Much of the previous work on protocol verification targets correctness. We target protocol performance and robustness in the presence of network failures. In addition, we provide new methods for studying multicast protocols and topology synthesis that previous works do not provide.

Network protocols are becoming more complex with the exponential growth of the Internet, and the introduction of new services at the network, transport and application levels. In particular, the advent of IP multicast and the MBone enabled applications ranging from multi-player games to distance learning and teleconferencing, among others. 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 [Pax97b, Pax97a]. Such failures are becoming more frequent, mainly due to the increased heterogeneity of technologies, interconnects and configuration of various network components. Due to the synergy and interaction between different network protocols and components, errors at one layer may lead to failures at other layers of the protocol stack. Furthermore, degraded performance of low level network protocols may have ripple effects on end-to-end protocols and applications.

To provide an effective solution to the above problems, we present a framework for the systematic design and testing of multicast protocols. The framework integrates test generation algorithms with simulation and implementation code. We propose a suite of practical methods and tools for automatic test generation for stacks of network protocols.

Thus far, we have investigated three approaches for test generation. The first is a heuristic approach that uses topological and event equivalence relations to reduce the problem space. The second approach, fault-independent test generation, uses a forward search algorithm to explore a subset of the protocol state space to generate the event tests automatically. State and fault equivalence relations are used in this approach to reduce the state space. The last approach, is fault-oriented test generation, that uses a mix of forward and backward search techniques to synthesize test events and topologies automatically.

We have also built a partial prototype of these methods in a network simulator, and applied it to two multicast routing protocols adopted by the Internet community; PIM-DM and PIM-SM. Our case studies revealed several design errors, for which we have formulated solutions with the aid of this systematic process.

We have further extended the system model to include end-to-end delays using the notion of virtual LAN. We use a variant of the fault-oriented test generation for performance evaluation of end-to-end multipoint protocols. We apply the new method to the timer suppression mechanism, a building block for several Internet multipoint protocols.

The rest of this document is organized as follows. Chapter 2.1 gives a brief overview of multicast, and chapter 2 presents related work in protocol verification, conformance testing and VLSI chip testing. Chapter 3 introduces the proposed framework, and system definition. We introduce the heuristic test generation in chapter 4. Chapters 5, 6, 7 present the search based approaches and problem complexity, the fault-independent test generation and the fault-oriented test generation, respectively. The application of our framework to end-to-end performance evaluation is presented in chapter 8. Chapter 9 concludes the dissertation by giving a summary of our contributions and presenting directions of future research.
Chapter 2

Related Work

The related work falls mainly in the field of multicast protocols, protocol verification and distributed algorithms. In addition, some concepts of our work were inspired by VLSI chip testing. Most of the literature on multicast protocol design addresses architecture, specification, and comparisons between different protocols. We are not aware of any other work to develop systematic methods for testing multicast protocol robustness.

There is a large body of literature dealing with verification of communication protocols. Protocol verification typically addresses well-defined properties, such as safety properties; freedom from deadlocks, and liveness properties; freedom from starvation.

In general, the two main approaches for protocol verification are theorem proving and reachability analysis (or model checking) [CW96, Hel].

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 and logic-based formalisms including first and higher order logic.

Reachability analysis algorithms [LCL87a, LCL87b] attempt 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 [Pro90, God90] could be used. These techniques focus only on parts of the state space and may use probabilistic [MS87], random [Wes86] or guided searches [PJ88].

In Section 2.2 we outline the main characteristics of protocol verification approaches, and discuss the adequacy of these approaches for the verification of multicast protocol robustness.

Work on distributed algorithms deals with synchronous networks, asynchronous shared memory and asynchronous networked systems [Lyn97]. Proofs can be established using an
automata-theoretic framework. Section 2.3 presents work on distributed algorithms, and outlines how it relates to our work.

Conformance testing is used to check that a given implementation of a protocol is equivalent to its specification. It does not target design errors or protocol performance, but implementation errors, and uses search techniques to attempt to cover the state space of the implementation. We discuss conformance testing in Section 2.4.

There is an analogy between our work and VLSI chip testing. Chip test generation methods attempt to generate test vectors to reveal faults in the VLSI fabrication process. These methods define a fault model and a circuit model for the chip under test and usually use search algorithms to find patterns exposing expected faults. The Built-In-Self-Test (BIST) [KBJND96] integrates test generation and fault detection algorithms in one scheme. VLSI chip testing schemes are discussed in Section 2.5.

Other related work includes a new approach for verification of cache coherence protocols [PD96]. This recent study shows how reachability analysis complexity can be reduced by using equivalence relations and symbolic representation of states. A global FSM (finite state machine) model was used to characterize the protocol behavior. One of our approaches (in Chapter 6) adopts some of the principles presented in the above study.

2.1 Multicast Protocols

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 dissertation, we address multicast protocols for the Internet, based on the IP multicast model. These protocols include multicast routing protocols (e.g. DVMRP [SD88], MOSPF [Moy92], PIM-DM [EFH+96a], CBT [BFC93], and PIM-SM [EFH+96b]), multicast transport protocols (e.g. SRM [FJL+96], RTP, and RTCP [SCFJ96]) and multiparty applications (e.g. WB [McC92], vat [JM93], vic [MJ95], nte [Han96a], and SDR [Han96b]), all of which we simply refer to as multipoint protocols.

2.1.1 Multicast Routing

The first part of this study focuses on multicast routing protocols, which deliver packets efficiently to group members by establishing distribution trees. Figure 2.1 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 conduct robustness case studies for PIM-DM and PIM-SM. 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.

---

\(^1\)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.
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 studies illustrate 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.

2.1.2 End-to-end Multipoint Protocols

The second part of this study targets performance evaluation of end-to-end multipoint protocols. We refer to multicast-based protocols as multipoint protocols. These include multicast transport protocols and multiparty applications. The design of multipoint protocols has introduced new challenges and problems. Some of the problems are common to a wide range of protocols and applications. One such problem is the multi-responder problem, where multiple members of a group may respond (almost) simultaneously to an event, which may cause a flood of messages throughout the network, and in turn may lead, for example, to unintended and counterproductive synchronized responses, and may cause additional overhead (e.g., the well-known Ack implosion problem), leading to performance degradation.

One common technique to alleviate the above problem is the multicast damping technique, which employs a timer suppression mechanism. We use this mechanism for our performance evaluation case study in Chapter 8.

This mechanism is employed in a range of multipoint protocols, such as: multicast routing protocols (e.g., PIM-DM [EFH+96a], PIM-SM [EFH+97] and IGMP [Fen97]), multicast transport protocols (e.g., SRM [FJL+96], RRM [GYE98], MFTP [MRTW98], and RTP/RTCP [SCFJ96]), address allocation (AAP [Han98], SDR [Han96b], and MASC [KRT+98]), adaptive web caching [ZMN+98], and other multipoint applications (media gateway [AMK98]).

2.2 Protocol Verification

Protocol verification is the problem of ensuring the logical consistency of the protocol specification, independent of any particular implementation. Protocol verification typically addresses safety, liveness, and responsiveness properties [SAASA95]. 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, which recovers the system to a legal state to resume normal execution from an illegal state. Most protocol verification systems aim to detect violations of these protocol properties.

Although we cannot do justice to the extensive body of work in this area, we shall dwell upon some of the main aspects and common approaches to protocol verification. There are two main approaches to protocol verification; theorem proving (using formal methods), and reachability analysis (sometimes called model checking).

2.2.1 Theorem Proving

In theorem proving, system properties are expressed in logic formulas, defining a set of axioms and rules. In contrast to reachability analysis and model checking, theorem proving can deal with infinite state spaces. However, interactive theorem provers require human intervention, and hence are slow and error-prone.

Theorem proving includes model-based and logic-based formalisms. Model-based formalisms, such as Z [Spi88], and Vienna Development Method (VDM) [Jon90], are suitable for protocol specifications in a succinct manner, but lack the tool support for effective proof of properties. The use of first order logic allows the use of theorem provers such as the Boyer-Moore logic prover (Nqthm [BM88]), but may result in specifications that are difficult to read. Higher order logic, such as Prototype Verification System (PVS) [ORS95], provides expressive power for clear descriptions, and proof capabilities for protocol properties.

In general, theorem proving systems require the definition of a set of axioms, and the construction of relations based on these axioms. The number of axioms and relations grows with the complexity of the protocol. These systems require strong mathematical background and understanding. The fact that axiomatization and proofs depend largely on human intelligence, may limit the use of theorem proving systems.

Theorem proving has been used in verification of distributed algorithms and systems, see Section 2.3.

Several attempts to apply formal verification to network protocols have been made. For example, assertional proof techniques were used to prove distance vector routing [Taj77], path vector routing [SC87] and route diffusion algorithms [JM82, GLA93] and [MS79] using communicating finite state machines.

An example point-to-point mobile application was proved using assertional reasoning in [RMP96] using UNITY [CM88]. Axiomatic reasoning was used in proving a simple
transmission protocol in [Hai85]. Algebraic systems based on the calculus of communicating systems (CCS) [Mil80] have been used to prove CSMA/CD [Par84]. Formal verification has been applied to TCP and T/TCP in [Smi96].

In all, formal verification methods may be important to protocol design. However, they have not been applied to wide-area multicast or complete routing protocols. We believe that theorem proving systems will be even more complex, and perhaps intractable, in the context of multicast protocols.

2.2.2 Reachability analysis

Most automated verification systems are based on exhaustive reachability analysis. To establish the observance of state invariants, it is sufficient to verify their correctness with a test for each state that is reachable from a given initial system state. The main problem that must be addressed in the design of such a system is the state space explosion problem.

Verification of state properties includes assertion violations and improper terminations. Verification of sequences of states includes non-progress conditions and temporal claims.

A reachability analysis algorithm attempts to generate and inspect all the states of a distributed system that are reachable from a given initial state. The three main types of reachability analysis algorithms are: 1) full search, 2) controlled partial search, and 3) random simulation.

If full search exceeds the memory or time limits, it effectively reduces to an uncontrolled partial search, and the quality of the analysis deteriorates quickly. Controlled partial search attempts to select a fraction of the full state space that can be searched within given time and space constraints. Random walk of the state space may be used for very large state spaces, where full or partial search is not feasible.

The typical measures of reachability analysis quality are:

- state coverage: the fraction of system states tested; i.e. \( \frac{\text{Number of Tested States}}{\text{Total Number of States}} \).
- error coverage: the fraction of system errors found. This measure represents the ability to find errors, and is not easily quantified, since the total number of errors present is usually unknown.

In practice, however, these measures may not be obtainable for complex protocols.
2.2.2.1 Full State Space Search

A finite state machine (FSM) is defined by a finite number of states and state transitions. Each state transition has a pre-condition and an effect or a post-condition. The transition is enabled only if the pre-condition holds. The effect of an execution can change the state of the system.

A reachable state, or sequence of states, can be checked for general safety conditions (e.g. absence of deadlocks, or buffer overruns), or protocol-specific requirements (e.g. temporal claim about a retransmission discipline).

States are stored and retrieved from a working set $W$. The algorithm performs a breadth-first (BF) or a depth-first (DF) search of the state space tree. BF finds the shortest error sequences first. DF requires a smaller work set $W$, in general. The depth of the search tree depends on the maximum length of a unique execution sequence. The width of the tree, on the other hand, is determined by maximum number of distinct execution sequence, usually a much larger number. For example, a protocol with 2 successors for every state, after $n$ transitions, the breadth is $2^n$ states, while depth is only $n$ states.

In DF, when an error is discovered, an execution sequence leading to the error may be easily produced. For BF, however, the execution sequence path must be reconstructed.

2.2.2.2 Controlled Partial Search

Controlled Partial Search is based on the premise that in most cases of practical interest the maximum number of states that can be analyzed, $A$, is only a fraction of the total number of reachable states $R$. Objectives of controlled partial search are to analyze precisely $A$ states, with $A = M/S$ (where $M$ is the memory available, and $S$ is the memory required to store one system state), such that: a) all major protocol functions are tested, and b) the search quality (i.e. the probability of finding any given error) is better than the coverage $A/R$.

Some controlled partial searches are based on:

1. depth-bounds: bounds are placed on the length of the execution sequences that are analyzed, limiting the search to a useful subset of behaviors, ruling out, degenerate cases of multiple overlapping sequences.

2. scatter-search: executions that lead closer to potential error states are selected. For deadlock, for example, an algorithm favors receive operations over send operations,
since one of the requisites of a deadlock is that all channels are empty. This may increase the probability of finding errors fast.

3. guided-search: the selection criterion is a cost function that is dynamically evaluated for each successor state. Not much has been proven about how useful a cost function is.

4. probabilistic search: successor states are explored in decreasing order of their probability of occurrence. Transitions in the system are tagged by probability of occurrence, and these are used as the selection criterion.

5. partial orders: the main factor responsible for the state space explosion problem is the large number of possible interleavings of concurrent events. Not all interleavings are necessarily relevant in the search for error states. The goal is to ‘prune’ away that part of the search proven to be irrelevant or redundant. One approach to achieve this is the formal definition of ‘equivalence’ relations on system behavior.

6. random selections: simplest, and may satisfy the objectives of controlled partial search.

The first 4 methods try to predict where the errors in a protocol can be found, which may be inherently risky, since one purpose of automatic verification is to capture unpredictable errors. Partial orders and random selection of successor states, in principle, avoid that problem. For partial orders, it is not trivial to prove irrelevance. For example if process A interacts with process C and B with C, there may be an implicit interaction between A and B. One cannot assume that A and B are disjoint and that all possible interleavings of their behaviors are necessarily equivalent.

2.2.2.3 Random Simulation

Random simulation may be used for huge problem sizes, where the memory requirements are larger than the available memory. This approach discards sets $A$ (the analyzed set of states) and $W$ (the working set), and explores the state space with random simulation or random walk. The quality of the algorithm in this case cannot be directly measured and the state coverage depends on the time given for simulation.

Recently, several researchers developed approaches to tackle the state explosion problem in a more uniform manner. We mention here fair reachability [LM96], and leaping reachability
In both cases, the protocol may be represented by communicating finite state machines (CFSM).

In the fair reachability analysis, the state reduction is achieved by forcing the protocol to progress through fair execution sequences, and hence cutting down the redundancy of state exploration. However, the result given in the above study only applies to the class of cyclic protocols (i.e. that have only one input channel and one output channel for each process) whose logical correctness is decidable. Its extensibility to other protocol classes and other models of finite state machines is questionable.

Leaping reachability analysis forces multiple machines of the protocol to progress by the concurrent execution of transitions at global states, hence leaping through the state space. Again, the assumption underlying the study (such as FIFO queues) may be very restrictive for real protocols. Also, these approaches mainly target deadlocks and liveness properties, and do not address robustness aspects per se.

In our work, however, we adopt approaches extending reachability analysis for multicast protocols. Our fault-independent test generation method (in Chapter 6) is similar to controlled partial search, and uses reduction techniques based on equivalence relations.

2.3 Distributed Algorithms

There has been much work on distributed systems and algorithms. Distributed algorithms may be classified based on the interprocess communication method or the timing model [Lyn97]. Communication methods include accessing shared memory, message passing, or remote procedure calls. With respect to timing, systems can be synchronous, partially synchronous, or asynchronous. Synchronous systems communicate in perfect lock-step synchrony. In contrast, asynchronous systems take steps in arbitrary orders. In partially synchronous systems, processors have partial information about timing of events, e.g., using approximately synchronized clocks.

Several failure models were considered in some of the studies on distributed algorithms, including message loss or duplication, and processor failures, such as stop (or crash) failures, transient failures, or byzantine failures [LSP82], where failed processors behave arbitrarily.

However, for our target domain, as well as for most applications, it is sufficient to assume with a high probability that faulty systems will crash cleanly [CDK95], hence we do not consider byzantine failures.
In [Lyn97], distributed algorithms are treated in a formal framework, using automata-theoretic models and state machines, and sometimes presenting results in terms of set-theoretic mathematics. The formal framework is used to present proofs or impossibility results.

Verification and proof methods for distributed algorithms include invariant assertions and simulation relationships, and are generally proved using induction. An invariant assertion is a property that holds true for all reachable states of the system, while a simulation is a formal relation between an abstract solution of the problem and a detailed solution. Invariant and simulation mapping proofs may be checkable using theorem-provers, e.g., Larch theorem-prover [GG91].

Asynchronous network components can be modeled as an input/output automata (I/O automata). This model allows the composition of different components as an I/O automaton. The correctness of the composed automaton can be based on proofs of correctness of its components. To include clocks or timeouts the timed-automata models are used [MMT91].

[Lyn97] presents models for the simpler systems, i.e., synchronous networks and asynchronous shared memory algorithms. Then discusses transformations to permit algorithms developed for the simpler systems to run in the more complex asynchronous network model. The synchronizer transformation [Awe85] enables asynchronous network systems to simulate synchronous networks, but does not work in the presence of faults. Another technique enables asynchronous networks to simulate asynchronous shared memory, and a third technique uses logical times [Lam78]. The monitoring technique allows detection of stable properties of the algorithm, such as termination or deadlock, by producing global snapshots of the system state [FGL82, CL85].

The asynchronous network model includes process and channel I/O automata models. The channel could be point-to-point FIFO queue (also called send/receive channel), broadcast, or multicast, where only a set of systems receive the messages sent to the channel.

Internet multipoint protocols that we address in this study can be modeled as asynchronous networks, with the components as timed-automata, including failure models. In fact, the global finite state machine (GFSM) model used by our search algorithms is adopted from asynchronous shared memory systems (in specific, cache coherence algorithms [PD96]) and extended with various multicast and timing semantics.

The transitions of the I/O automaton is given in the form of pre-conditions and effects. This is similar to our representation of the transition table for the fault-oriented test
generation method. Also a cause function is used to describe the connection between message sending and receiving events. This is similar to one of our implication rules in Chapter 8.

Theorem proving methods can be used with distributed algorithms to prove safety properties (e.g., absence of deadlocks), liveness (e.g., absence of livelocks), or give impossibility results. It may also be used to establish asymptotic bounds on the complexity of the distributed algorithms.

It is not clear, however, how theorem proving techniques can be used in test synthesis to construct event sequences and topologies that stress network protocols. Also, aside from asymptotic behavior, it may be hard for such techniques alone to address performance issues.

In sum, we feel that parts of our work draw from distributed algorithms verification principles. Yet we feel that our work complements such work, as we focus on test synthesis problems. The combination of timed automata, invariants, simulation mappings, automaton composition, and temporal logic [Lam94] seem to be very useful tools for proving (or disproving) and reasoning about properties of network protocols.

2.4 Conformance Testing

A given implementation ideally realizes all functions of the specification, over the range of acceptable parameter values, and rejects erroneous inputs. A conformance test is used to check that the external behavior of a given implementation of a protocol is equivalent to its formal specification. A conformance test should fail only if implementation and specification differ. In contrast, verification of the protocol must always reveal the design error.

Given an implementation under test (IUT), sequences of input messages are provided and the resulting output is observed. The test passes only if all observed output matches those of the formal specification. Another approach of conformance testing is to establish the conformance of the control structure of the implementation to the structure of the specification. Implementation and specification have the same structure if they model equivalent sets of states and allow for the same state transitions.

A state of the IUT is a stable condition awaiting input signal. A transition is the consumption of an input signal, the possible generation of an output signal, and the possible move to a new state. The move must be deterministic in order for the test to be reproducible. In each state, a complete IUT can accept and respond to all input symbols
from the complete system vocabulary. The acceptance of an input signal that is outside the official input vocabulary may cause a transition into a set of states that produces erroneous behavior.

The series of input sequences used this way is called a conformance test suite. The cost of the test can be expressed as the length of the test suite, i.e. the total number of messages sent to the IUT. The main problem is to find an efficient procedure for generating a conformance test suite for a given protocol.

One possible solution is to generate a sequence of state transitions that passes through every state and every transition at least once; also known as a transition tour. The problem of finding a minimum length transition tour of a finite state machine, described for instance in [Kle80], can be solved in polynomial time.

However, in order for this solution to work, the state of the machine must be checked after each transition, since the implementation may be faulty. This leads to the definition of UIO sequences.

A Unique Input/Output (UIO) sequence or state signature is a sequence of transitions that can be used to determine the state of the IUT. To be able to verify every state in the IUT, we must be able to derive a UIO sequence for every state separately.

This approach generally suffers from the following drawbacks: a) Not all UIO sequences are necessarily different. It may be possible to derive a distinguishing sequence; a single UIO sequence that can be used to identify any state in a finite state machine (FSM). b) Not all FSMs have such a distinguishing sequence, and not all states have a UIO sequence. c) Even if all states in a FSM have a UIO sequence, the problem of deriving UIO sequences has been proved to be PSPACE-complete in [YL91]; i.e. only very short UIO sequences can be found in practice, and d) UIO sequences can identify states reliably only in a correct IUT. Their behavior for faulty IUTs is unpredictable, and they cannot guarantee that any type of fault in an IUT remains detectable. Only the presence of desirable behavior can be tested by conformance testing, not the absence of undesirable behavior.

In conclusion, conformance testing techniques are important for testing protocol implementations. However, it is not suitable, as is, to be used in the design stage of a protocol. We consider work in this area as complementary to the focus of our study.

2.5 VLSI Chip Testing

Chip testing uses a set of well-established approaches to generate test vector patterns, generally for detecting physical defects in the VLSI fabrication process.
Common test vector generation methods detect single-stuck faults; where the value of a line in the circuit is always at logic ‘1’ or ‘0’. Test vectors are generated based on a model of the circuit and a given fault model. The cost of the test generation depends on the complexity of the circuit to be tested, as well as the method of test generation. Random vector generation is simple, but in general performs poorly—in terms of fault coverage—if the vector set is not large. In contrast, deterministic vector generation produces shorter and higher quality tests by processing a model of the circuit, and hence is more expensive. Deterministic vector generation can be fault-independent or fault-oriented. In a fault-oriented process, test vectors are generated for specified faults as defined by the fault model. On the other hand, a fault-independent process works without targeting individual faults.

In the fault-oriented process, the two fundamental steps in generating a test vector are: a) to activate (or excite) the fault, and b) to propagate the resulting error to an observable output. Activating a fault involves a line justification step; that is, setting circuit input values to cause a line $l$ in the circuit to have a specific value. To propagate the error to an output, a path from $l$ to the output needs to be sensitized. A line whose value in the test $t$ changes in the presence of the fault $f$ is said to be sensitized to the fault $f$ by the test $t$. A path composed of sensitized lines is called a sensitized path. Several algorithms have been developed to solve the path sensitization problem, such as the D-algorithm, the 9-V algorithm, and the Path-Oriented Decision Making (PODEM) algorithm.

Line justification or error propagation usually involve a search procedure with a backtracking strategy to resolve or undo contradiction in the assignment of line and input values. The line assignments performed sometimes determine or imply other line assignments. The process of computing the line values to be consistent with previously determined values is referred to as implication. 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.

Fault-independent test generation attempts to generate a set of input vectors that detect a large set of faults without targeting individual faults. One such method is the critical-path method. The basic steps of a critical-path algorithm is to a) select an output and assign it to a value, then b) recursively justify the value of a gate output by assigning values to the gate input.

Another concept of VLSI testing in which we are interested, is fault equivalence. Two faults $f$ and $g$ are said to be functionally equivalent for a circuit $C$ under test $x$ iff $C_f(x) = C_g(x)$. A test $t$ is said to distinguish between two faults $f$ and $g$ if $C_f(t) \neq C_g(t)$; such
faults are distinguishable. The relation of functional equivalence partitions the set of faults into equivalence classes. For fault analysis it suffices to consider only one fault from every equivalent class.

A scheme that utilizes the above concepts for on-line chip testing is the Built-In-Self-Test (BIST) [KBJND96]. BIST provides a systematic technique for chip testing synthesis. A generic BIST scheme is shown in figure 2.2.

![Generic BIST scheme](image)

This technique can be used to detect faults due to single-stuck-line. BIST uses a test pattern generator (TPG) to produce the input patterns applied to the circuit under test. The test patterns are chosen to maximize fault coverage with a minimum number of inputs. A ‘response monitor circuit’ is used to monitor and detect error signals. The expected output for VLSI chip testing is ‘fault coverage vs. test length’ curve.

We are particularly interested in the architectural paradigm of BIST, after which we model our simulation method. Our approaches for protocol testing use some of the above principles; such as forward and backward implication, fault-independent and fault-oriented approaches.

However, in VLSI chip testing, the test vectors are produced for a given circuit, whereas in protocol test generation the topology is variable, and a protocol should be designed to work with arbitrary topologies, which adds another dimension to our problem.
Chapter 3

Framework Overview

Protocols may be evaluated for correctness or performance. We refer to correctness studies that are conducted in the absence of network failures as verification. In contrast, robustness studies consider the presence of network failures (such as packet loss or crashes). In general, the robustness of a protocol is its ability to respond correctly in the face of network component failures and packet loss. This dissertation presents a methodology for studying and evaluating multicast protocols, specifically addressing robustness and performance issues. Supported by a set of tools for automatic test generation and synthesis, the method integrates protocol modeling, simulation and implementation in a single framework. The major contribution of this work lies in developing new methods for generating stress test scenarios that target robustness and correctness violation, or worst case performance.

We adopt a system’s approach to failure and behavioral analysis. That is, instead of studying protocol behavior in isolation, we incorporate multiple protocol layers with network dynamics and failures in order to reveal more realistic behavior of protocols in operation.

This chapter presents an overview of the framework and its constituent components. The model used to represent the protocol and the system is presented along with definitions of the terms used.

3.1 Framework Overview

Our framework integrates test generation with simulation and implementation code. It is used for Systematic Testing of Robustness by Evaluation of Synthesized Scenarios (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. Scenarios will be discussed in more detail in Chapter 4.
The input to this framework is the specification of a protocol, and a definition of its design requirements, in terms of correctness or performance. Usually robustness is defined in terms of network dynamics or fault models. A fault model represents various component faults; such as packet loss, corruption, re-ordering, or machine crashes. The desired output is a set of test-suites that stress the protocol mechanisms according to the robustness criteria.

The STRESS framework includes the following components (see figure 3.1):
1. Automatic test generation and topology synthesis algorithms,
2. Detailed simulator driven by the synthesized test patterns and scenarios, and
3. Protocol implementation driven through an emulation interface to the simulator.

### 3.1.1 Test Generation

The core contribution of our work lies in the development of systematic test generation algorithms for protocol robustness. We investigate three such algorithms, each using a different approach.

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 3.2.3 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.

Heuristics may be developed, however, to increase the test quality. Our first approach utilizes topological and event equivalences to establish an initial set of tests. This set is then expanded to include the possible fault scenarios (e.g., message loss), according to the protocol robustness. We use a simulation-based method to run the tests, analyze erroneous behavior and collect coverage information. The heuristic approach and the simulation method are described in Chapter 4.

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. 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. Chapter 6 describes our fault-independent approach.

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 present our fault-oriented algorithm in Chapter 7.

In Chapter 8, we further extend the fault-oriented algorithm to address end-to-end performance evaluations by synthesizing worst and best case performance scenarios.

We will present these algorithms in more detail in later chapters of this document, along with several case studies. The case studies are applied to PIM-DM to illustrate differences between the approaches, and provide a basis for comparison. In addition we apply the heuristic approach to PIM-SM to illustrate how test generation can be applied to different flavors of multicast routing. For end-to-end performance evaluation, we apply our method to the timer suppression mechanism in Chapter 8.
3.1.2 Detailed Simulation

Usually, automatic test generation is performed on a protocol model, that sometimes abstracts out some characteristics of the protocol. An error that may be experienced in the abstract model, may not be experienced in a more detailed model of the protocol, such as a detailed simulation, and vice versa. For this reason, the test sequences generated from the abstract model, are further validated by driving a simulator and analyzing the output.

Also, the first algorithm based on heuristics, uses the simulator as an integrated part to generate the tests. In a later stage, these tests are applied to the implementation code.

We have implemented detailed simulators for PIM-DM and PIM-SM in the network simulator (NS), and used them for parts of our case studies. Chapter 4 describes how the simulation environment is integrated with the test generation. The simulations of SRM in NS can also be conducted using the scenarios generated for the timer suppression mechanism.

3.1.3 Implementation Interface

An emulation interface to the simulator may be used in order to observe how the actual implementation of the protocol behaves under the generated tests.

This will enable us to: a) conduct conformance tests, by applying conformance test-suites through the emulator, and b) perform thorough analysis of the correctness and performance of the protocol implementation under test.

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

3.2 System Model and Definition

3.2.1 The system model

We define our target system in terms of network and topology elements and a fault model.

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

For end-to-end performance evaluation, the multicast distribution tree is abstracted out as delays between end systems and patterns of loss for the multicast messages. Cascade of LANs or uniform topologies are addressed in future research.
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. Note that a fault may not necessarily be an error for the low level protocol.

The fault model may include:

- Loss of packets, such as 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. We take into consideration selective packet loss, where a multicast packet may be received by some members of the group but not others.

- 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 our analyses are addressing. Sometimes delay fault problems may be translated into event sequencing problems, as we will show by example in Section 7.3.3. For end-to-end delays, the delays incurred by the network are those of the multicast distribution tree and depend upon the multicast routing protocol in addition to the above mentioned network factors.

- 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. For our robustness studies we study PIM. The designing robustness goal for PIM is to be able to recover gracefully (i.e. without going into erroneous stable states) from single protocol message loss. 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. In addition, we study PIM protocol behavior in presence of crashes and route inconsistencies. For end-to-end studies, we consider extended delays and selective packet loss among group members.
3.2.2 Test Sequence Definition

A fault model may include a single fault or multiple faults. For performance studies, the model may include multiple faults, such as extended delays and packet loss patterns that lead to degradation of protocol performance (see Chapter 8). In our robustness studies, however, we adopt a single-fault model, where only a single fault may occur during a scenario or 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 final \( P(q, T') \) is incorrect; i.e. the stable state reached after the occurrence of the fault does not satisfy the protocol correctness conditions (see Section 3.2.5) irrespective of \( P(q, T) \).

In case of a fault-free sequence, where \( T = T' \), the error is attributed to a protocol design error. Whereas when \( T \neq T' \), and final \( P(q, T) \) is correct, the error is manifested by the fault. This definition ignores transient protocol behavior. We are only concerned with the stable (i.e. non-transient) behavior of a protocol.

3.2.3 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 >’, as shown in figure 3.2.

![Test pattern dimensions](image)

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

\(^1\)This facilitates the analysis of erroneous behavior.
• Topology: $T = < N, L >$ is the routed topology of set of nodes $N$ and links $L$. $N = < n_1, n_2, ..., n_k >$, is the list of nodes each running a set of protocols. $L = < l_1, l_2, ..., l_m >$ are the links connecting the nodes; two in case of a point-to-point link, or more for LANs. A link has a delay and a bandwidth. This model is extended to represent various delays and bandwidths between pairs of nodes, by using a virtual LAN matrix (see Chapter 8).

• 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.

3.2.4 Test requirement

• **reachability**: the test should drive the protocol into erroneous states reachable from a given initial state.

• **controllability**, or controlled fault model: the test should not introduce additional faults except those specified by the fault model. For example, no extra loss should occur due to queue overflows. This may be realized in a simulator by using virtually infinite queue lengths.

• **observability**, or error propagation: unless otherwise specified by the protocol, a data packet that is lost (duplicated) in a LAN topology is not re-produced (absorbed) by the network, and hence can be observed by the end-points.

The reachability requirement is general, while controllability and observability mainly deal with the simulation environment (e.g., similar to that used in Chapter 4). For the finite state machine models we assume full controllability and observability.

3.2.5 Brief description of PIM-DM

For our robustness studies, we apply our automatic test generation algorithms to a version of the Protocol Independent Multicast-Dense Mode, or PIM-DM. The description given here is useful for Chapters 4 through 7.

---

2 This is a reasonable assumption since our focus is on testing protocol design, not implementation.

3 The heuristic test generation was also applied to PIM-Sparse Mode (see appendix A). We use the term PIM to indicate both PIM-DM and PIM-SM.
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 in general, is to deliver data from senders to group members (only those that have joined the group) without any data loss. For our methods, we only assume that a correctness definition is given by the protocol designer or specification. For illustration, we discuss the protocol errors and the correctness conditions.

### 3.2.5.1 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. The protocol errors are defined in terms of the end-to-end behavior, and may be used to capture the error in a simulation environment, where the end-point traces may be obtained (for example, see Chapter 4). 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**: lack of packet delivery after a receiver joins the group.
5. **leave latency**: unnecessary packet delivery after a receiver leaves the group.

---

4Join and leave latencies may be considered in other contexts as performance issues. However, in our study we treat them as errors.
6. \textit{wasted bandwidth}: unnecessary packet delivery to network links that do not lead to group members.

3.2.5.2 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\(^5\).

   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.

The mapping from functional correctness requirements for multicast routing in general (e.g., single delivery of packets to group members), to the definition in terms of the protocol model (e.g., if there is a member then there exists exactly one forwarder) is currently done by the designer. The automation of this process is part of future research.

\(^5\) 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.
Chapter 4

Heuristic Test Generation

In this chapter we present one test generation algorithm based on a heuristic, topology equivalence relations, and simulation.

After giving an overview of the approach, we illustrate how it can be applied to multicast routing protocols by conducting two case studies on the Protocol Independent Multicast (PIM).

4.1 Method Overview

The main purpose of this approach is to identify a set of scenarios that may experience a protocol error in the presence of single message loss. These scenarios are chosen from a set of representative scenarios and topologies obtained through equivalence relations and heuristics. The scenarios are then simulated and the output is analyzed to identify errors.

The simulation method consists of three stages: scenario generation (pre-processing), tracing (simulation), and output analysis (post-processing). Figure 4.1 illustrates these stages. The building blocks in the figure are explained in detail throughout the rest of this section.

Note that the engineering design process is usually iterative, where an investigator may cycle and feedback into previous stages based on his/her intuition and insight, sometimes gained by the analysis of earlier simulations. Our methodology does not contradict such process. In fact we will show in appendix A how we iterated through the stages to guide our simulations. The following section, however, only discusses the modules supporting the different stages.
4.1.1 Scenario Generation

Scenarios are composed of routed topologies and sequences of events (input stimuli and state transitions), and describe the simulation context that may cause protocol transitions. Scenario parameters include the routed topology, host scenarios and loss scenarios.

4.1.1.1 Routed topology

The routed topology is the network infrastructure upon which the protocol operates; nodes, links and low level protocols, e.g. unicast routing.

We try to identify simple topologies that facilitate the evaluation of the main mechanisms of the protocol, and to which other, more complex topologies may be reduced\textsuperscript{1}. We choose a LAN with four connected routers as the basic topology. We show how other topologies are reducible to the four-router LAN topology, and discuss the limitations of such a topology in section 4.2.1. We further extend the topology to capture particular characteristics of the protocol under study, PIM.

![Figure 4.1: The block diagram of the simulation method](image)

As a component of the routed topology, unicast route inconsistencies may be a common source of error. Unicast routing may exist in one of the following three states: (a) consistent routing, (b) transient inconsistent routing, and (c) long lived inconsistency. Case (a) requires no changes. The study of case (b) is convergence analysis, which we do not address here. We are particularly interested in case (c)\textsuperscript{2}. We add an inconsistent unicast

\textsuperscript{1}Two topologies are said to be reducible (or equivalent) if they drive the protocol (according to the host scenarios applied) into the same states, experiencing the same set of state transitions.

\textsuperscript{2}This may be caused by a multicast region spanning more than one unicast routing AS.
routing component to force the multicast routing protocol into states encountered in such pathology, and analyze those states.

4.1.1.2 Host scenarios

Host scenarios are combinations of possible host actions. In our case study, these are defined by the multicast service model. Host actions include joining (or leaving) groups, or sending packets to groups. For large numbers of hosts and groups it is prohibitively costly to explore all possible combinations exhaustively.

The heuristics used in this study do not guarantee that all faulty scenarios for a protocol will be covered. Our more practical and achievable objective is to study multicast protocol behavior for scenarios that include the primary host events; in this case, joining a group, leaving a group and sending to a group. For these scenarios, we generate all possible message loss cases and extract the faulty scenarios automatically.

We choose a simple multicast host scenario that has a single source ‘S’ and two receivers ‘R1’ and ‘R2’ for the same group.

We estimate all the possible combinations of our host model, and try to reduce the number to those simple scenarios that support the main protocol functions. We call such scenarios representative scenarios. To obtain the representative scenarios we apply the scenario filter shown in figure 4.2.

![Figure 4.2: The scenario filter](image)

The use of the filter shown in the figure is illustrated by the following example. For one source and two receivers, the five possible host events are: source ‘S’ sending to a group (or ‘S’ for short), receiver joining a group (or ‘J1’ and ‘J2’ for receivers ‘R1’ and ‘R2’, respectively), and receiver leaving a group (or ‘L1’ and ‘L2’ for receivers ‘R1’ and ‘R2’, respectively).

For all possible permutations, there exists $5! = 120$ scenarios, considering that each host event occurs once. Then, as shown by figure 4.2, we apply protocol constraints,
(e.g. a receiver does not leave before it joins the group), to reduce the number of possible combinations to $5!/(2! \times 2!) = 30$ scenarios. Further, as a practical input, we assume, without loss of generality, that (the source sends packets throughout the simulation), to reduce the number of possible scenarios to $30/5 = 6$ scenarios. These six scenarios are:


The number of representative scenarios can be further reduced if the host distribution is symmetric with respect to the topology, since the following scenarios will be equivalent: (i) 1 equivalent to 5, (ii) 2 equivalent to 4, and (iii) 3 equivalent to 6; i.e. we need only investigate 3 different host scenarios for the given topology.

### 4.1.1.3 Loss and Failures

Are defined by the fault model; the single message loss. This model includes ‘selective loss’, where a message sent on a LAN may be lost by any of the intended receivers. The input to the ‘loss & failures’ substage (shown in figure 4.1) is obtained from initial traces of simulations without protocol message loss. These traces guide further simulations to cover all possible protocol message loss scenarios.

### 4.1.2 Simulation and Tracing

During this stage the protocol mechanisms are simulated and traces are collected:

**Simulation**  One desirable approach for simulating complex protocols, is to include detailed mechanisms of parts of the protocol while abstracting out others, we call this approach subsetting. To maintain protocol correctness, however, an abstracted part must be replaced by its equivalent that exhibits similar external behavior, under the study assumptions. Subsetting allows us to focus on specific parts of the protocol state space, and can be based on protocol functions, states or messages. Subsetting protocol functions or mechanisms refers to the abstraction of these functions. This may be achieved by replacing a complex mechanism by a simpler one, exhibiting similar external behavior under relaxed assumptions. For example, we may use static configuration instead of simulating a detailed bootstrap algorithm. This way, we may study other protocol mechanisms assuming correctness of the bootstrap mechanism. Using protocol states subsetting, a study may focus on specific protocol states. This allows, for example, the study of multicast group
state without dealing with source-specific state. Subsetting protocol messages allows the examination of specific protocol message types in the absence of others.

**Tracing**  Tracing is the process of logging information about events or packets during the simulation run. Logged information is analyzed during the post-processing (i.e. the output analysis) stage. In addition, some traces are used as feedback to the scenario generator to guide further simulations. We consider several kinds of tracing:

**End-point tracing:** Tracing end-points includes logging information pertaining to hosts sending or receiving packets, and joining or leaving multicast groups. A detailed description of the traces is given in the case study sections.

To identify errors and pathologies in the protocol itself, we focus on the effect of the multicast routing protocol transitions on the end-point packet delivery (as explained in section 4.1.3).

**Protocol transition tracing:** A protocol can be represented by a finite state machine (automaton), consisting of states, transitions and stimuli (inputs, outputs, and timer actions). Based on knowledge of initial protocol states, we obtain the sequence of protocol transitions by tracing all stimuli.

We use protocol traces to diagnose and verify protocol behavior, and to analyze errors.

**Link tracing:** We keep track of packets traversing links, as well as events of packet loss on links. Link tracing is mainly used for fault injection; links carrying message types of interest are targeted for intentional message loss in further simulations. We achieve this through feedback to the scenario generation stage, as shown in figure 4.1. This reduces the number of loss scenarios examined to those directly affecting the protocol behavior under investigation. We also use link tracing in output analysis and visualization.

**Code annotation:** When placed in key points (such as beginning of protocol procedures or code modifying the state of the protocol), code annotations capture internal execution of the protocol machinery. We use code annotation to estimate what part of the code, and subsequently the protocol, has been executed and stressed (code coverage).
4.1.3 Output Analysis

One major concern of our approach is to identify pathological cases, and indicate when and if an error occurred and why. This is achieved in the output analysis stage, which consists of:

**Identifying end-point errors** Error conditions may be specified with respect to end-point traces as mentioned in section 3.2.5. If the factors during one simulation run are relatively static (i.e. static unicast routing, static topology and controlled loss), the end-point error may be attributed to an error in the multicast routing protocol.

Once the specified error is identified by the output analyzer, the trace log is rolled back in time to investigate the protocol traces, as explained next.

**Relating errors to protocol** After detecting an end-point error, the output analyzer isolates possible causes of such errors in the form of protocol traces\(^3\). The output analyzer in this case is similar to a logic analyzer, allowing the designer to navigate backward in time and investigate the causes of the error.

As will be shown in the case studies, the process of identifying a protocol error may suggest fixes to the problem.

**Code profiling** The profiler captures information about the annotated code, such as which procedures were (or were not) invoked, and the order and frequency of invocation. This information indicates the portion of the protocol stressed by the examined scenarios.

4.2 Case Study

To evaluate the utility of the heuristic approach, we applied it to a complex multicast routing protocol; PIM. Both PIM-SM and PIM-DM are considered in the case study. However, we present the PIM-DM study briefly in this section, and present the details of the PIM-SM in appendix A.

Being robust to (at least) a single message loss, even in the presence of unicast inconsistencies, was a design goal for PIM, as was described in section 3.2.1. The PIM-DM protocol was described in section 3.2.5.

\(^3\)In our experience, the cause for end-point errors was often due to protocol misbehavior in the recent trace history of the error.
We point out two PIM mechanisms relevant to this study; *Assert* and *prune-override*. The PIM *Assert* mechanism is the process by which at most one forwarder for a LAN is selected to avoid duplicates in case of multiple potential forwarders due to parallel paths to the source. The *prune-override* enables a downstream router (i.e. with downstream members) to retain its established branch of the tree, in case another router on the same LAN tries to prune that branch\(^4\).

The rest of this section is outlined as follows. Section 4.2.1 establishes the equivalence relationship for the topology used for the case study. Section 4.2.2 describes the simulation test suites, and section 4.3 presents an example for applying the method.

### 4.2.1 Topology Equivalence

Two topologies are equivalent if they drive the protocol transitions into the same states, under the same set of event sequences. A topology is reducible to another topology –with fewer connections and routers– if the two topologies are equivalent.

We show in this section that for single message loss scenarios, the *four-router* LAN topology adopted in this study experiences the same protocol errors that an *N-router* LAN topology experiences –where \(N > 4\)–, and hence they are equivalent, for PIM *joins*, *prunes* and *asserts*. For brevity, we only prove equivalence in the case of *prune* messages, and hint to the proof approach in the other cases. We also identify assumptions and limitations under which this equivalence relationship holds.

---

\(^4\)To achieve this, a downstream router receiving a *prune* on its incoming interface triggers a *join* upstream.
Prunes: First, we consider $N$-router LAN topologies, where $N = 1, 2, \text{and } 3$, respectively. It is trivial to prove that these topologies are not equivalent for hop-by-hop messages.

Assumption $N$-router LAN topology, where $N > 3$, is reducible to the three-router LAN topology for prunes, w.r.t. single message loss scenarios.

To justify our assumption we first prove that a four-router LAN topology is reducible to a three-router LAN topology.

Correctness condition: As described by section 3.2.5, the conditions necessary to avoid packet duplication and black holes may be stated as \footnote{For brevity we only consider black holes and packet duplicates correctness conditions.}: If a router on the LAN has the LAN as its incoming interface, there must be one other router with the LAN in its outgoing list. Once this condition is satisfied, violating it is considered a protocol error\footnote{This is to differentiate between join latency (which is not considered a protocol error) and a black hole which is a protocol error.}.

Next, we examine the three-router LAN topology. In figure 4.3, topology 'T', assume that $A$ and $B$ are downstream routers, and $C$ is an upstream router.

- In figure 4.3, topology 'T', router $C$ has the LAN in its outgoing list, router $A$ has the LAN as its incoming interface, and router $B$ is leaving the group and so sends a prune towards $C$. The prune is multicast on the LAN.

The only case where the correctness condition may be violated is when $C$ receives the prune while $A$ does not. In the other cases, either the prune is not received by $C$, or is received by $A$ which triggers a prune-override to re-establish the LAN in $C$’s outgoing list. This is illustrated by the following selective loss pattern table for the prune message sent by $B$:

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

where a ‘0’ indicates no-loss and ‘1’ indicates loss. The error occurs where the upstream router ($C$) received the prune, but the router with downstream members ($A$) did not receive it.
• In figure 4.3, topology ‘Π’, we add another downstream router $D$. The selective loss pattern table follows:

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The only error occurs when the upstream router ($C$) receives the prune, but neither of the downstream routers receives it. If the prune is received by any of the downstream routers, a prune-override would re-establish the LAN in $C$’s outgoing list.

From the symmetry of the loss patterns and topology we see that all errors are triggered by the same transitions experienced by router $A$ in topology ‘Γ’. Hence, the extended topology ‘Π’ does not introduce any new errors, and exhibits the same external behavior as does topology ‘Γ’. We conclude that topology ‘Γ’ and topology ‘Π’ are equivalent for prunes.

We now show that the $N+1$-router LAN topology is reducible to the $N$ case; where $N \geq 3$.

With the addition of an upstream router (figure 4.3, topology ‘ΠΠ’), no added error cases are encountered. The addition of a downstream router, however, may introduce new error scenarios. Similar to the four-router LAN case, we establish the following assertion: ‘the only error case occurs when all downstream routers lose the prune and the upstream router receives it.’ If the prune was received by any of the downstream routers, the correctness condition would be retained using prune overrides$^7$.

The assertion holds in both topologies. Hence, we conclude that the $N$-router topology experiences the same errors as the $N+1$-router topology.

---

$^7$Prune-overrides are actually join messages. The effect of join message loss is described in section 4.2.1.
Figure 4.4: The topology used for the case study

From the above we see that by simulating the three-router LAN topology we capture all the errors, with respect to selective loss (for the prune mechanism), that may be experienced by any N-router LAN topology; where \( N > 3 \).

**Joins** and prune-overrides: For prune-overrides the only router affected by the message is the destined upstream router, hence the equivalence is readily established. The loss of a PIM-SM join may lead to join latency, but does not cause black-holes. Joins leading to packet duplication lead to asserts that are discussed next.

**Asserts**: In most cases, proofs similar to the prune case can be applied to Asserts. However, since asserts may be triggered due to parallel paths, the base case is established for the four-router LAN topology. Figure 4.3, topology 'III', represents the four-router LAN topology, where \( A \) and \( B \) are downstream routers, and \( C \) and \( D \) are upstream routers\(^8\).

---

\(^8\)For PIM-SM, a limitation to the four-router LAN topology is given for the esoteric case of three upstream routers and three downstream routers with inconsistent unicast routing tables. This case creates one extra transition that can only be captured by at least a six-router LAN topology. We do not consider this a practically significant scenario, and we consider its analysis as a special case, not captured by the four-router LAN topology. However, aside from this exception, the \( N \)-router LAN topology (where \( N > 4 \)) is equivalent to a four-router LAN topology w.r.t. asserts.
For our case study, we use a *four-router* LAN topology with an added source S1. The overall physical topology consists of five routers, four of which are connected via a LAN, as shown in figure 4.4.

### 4.2.2 Test suites

In this section we elaborate on the routed topology, host scenarios and loss pattern generation used for our case study. We also describe the simplifications and subsettings applied.

**Physical and routed topologies**  The overall topology used is that shown in figure 4.4. For the unicast routing protocol we use a centralized version of Dijkstra’s Shortest Path First (SPF) algorithm [Dij59].

PIM uses the underlying unicast routing tables for building multicast trees. Therefore, unicast routing inconsistencies affect the operation of PIM. To investigate such interaction we add a component to force inconsistent multicast routes between PIM routers, as shown in figure 4.4, topology 1.

**Host scenarios**  Since protocol states for different groups do not interact, we consider only one group. Also, since protocol states for different sources do not interact, it suffices to consider only one source ‘S’ per simulation run. The source is modeled as a constant-bit rate (CBR) stream with fixed packet size. The source model does not affect the correctness of the method. However, to assure full controllability over the selective loss model, we set the data rate to ensure that no loss occurs due to queue overflow.

While we consider only a single source, we consider two receivers (‘R1’ and ‘R2’) for the same group to account for shared tree state interactions. We use the representative host scenarios described in section 4.1.1.2.

**Loss patterns**  We investigate all possible selective loss scenarios for multicast hop-by-hop PIM messages in the equivalent topology.

Loss models are applied exhaustively to those links that carry protocol messages under investigation. The tracing stage identifies these links during the first simulation run, without packet loss, and feeds back the link information to the loss generation module, as

---

9 We do not consider aggregated source or group entries in this study.

10 For this we use packet size of 180 bytes, and a send interval of 25 ms (i.e. source rate of 57.6 kb/s), this ensures no queue drops on the 1.5 Mb/s links used with 10 packet queue limit.
shown in figure 4.1. As we will show in section A.4, the number of representative scenarios is quite small, and hence the number of overall lossy scenarios explored is manageable.

**Tracing** Trace information includes the event type (send or receive), the node experiencing the event, the type of message sent or received, and the time at which the event occurred. Every data packet is assigned a unique sequence number.

### 4.3 Applying the Method

This section provides an illustrative example, showing how the heuristic approach may be used to identify and analyze errors encountered during the simulation of the representative scenarios.

We have implemented an initial version of the method in the Network Simulator ‘NS’ [MF95]. NS is an event-driven packet-level simulator controlled and configured via Tcl [Ous94] and Object-Tcl (or OTcl [WL95])\(^\text{11}\). To support our method, we have added modules to provide LAN support, controlled selective loss, protocol tracing, profiling capabilities, and a detailed implementation of PIM-DM and PIM-SM. This implementation serves as the simulation environment for our case study. In addition, the building blocks were designed to be re-used within the same framework to apply this method to other multicast protocols.

To verify that our implementation conforms to the protocol specification, we ran several conformance test-suites using the simulator.

#### 4.3.1 Obtaining faulty scenarios

To obtain the *faulty scenarios* (i.e. those that contain errors), we execute the method stages in order (i.e. scenario generation, simulation and tracing, and output analysis, respectively), and then reverse the order from the output to the traces to identify the faulty scenarios. These phases are automated by the tools provided, and are transparent to the user, once the scenario set-up is complete.

The process of attributing end-point errors to protocol actions may be automated only if the error conditions are given in terms of such protocol actions. In practice, these protocol error conditions are often not known *a priori* by the designer, and are usually defined in terms of end-point errors (such as packet loss or duplication). The supporting

\(^\text{11}\)For information about the simulator see http://catarina.usc.edu/vint.
tools identify end-point errors, and provide a history of protocol traces. The designer then examines the traces and identifies the protocol errors. This process may suggest fixes to the problem, as we will show in section A.

4.3.2 Example for PIM-DM

In this section, we briefly describe one example in which the heuristic approach was used in conjunction with simulation to reveal design errors in PIM-DM. Other examples and results that were obtained using this approach are provided in appendix A. The scenario presented here was identified after the simulation and analysis of the representative scenarios with the selective loss model.

We used the representative scenario ‘J1;J2;L1;L2’ over topology 1. In this scenario the fault was represented by the loss of the join (i.e. prune – override) message sent by router A, as shown in figure 4.5.

The error in this scenario was observed as a gap in the sequence number of the packets received by receiver R2, indicating a blackhole. The start of this gap was synchronized with the event ‘L1’, i.e. the leave of receiver R1, when the router B triggered a prune onto the LAN. The failure of router A to override this prune caused the blackhole. This failure was caused by the loss of a single join message, and hence the robustness requirement for PIM-DM was not satisfied.

To fix this problem, a second chance should be given to the downstream router (in this case A) to override the prune. This may be achieved, for example, by having the leaving
router (B in this case) send two prunes when leaving, or have the upstream router (C) send a prune — alert in the form of a prune on the LAN before removing the LAN from its routing entries.

Although the heuristic approach may capture some design errors, it lacks formality and does not produce the test-suites automatically. We attempt to address these issues in the other two approaches; the fault-independent and fault-oriented test generation.
Chapter 5

Search-based Approaches

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 (either correctness or performance), 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. In this chapter we present two such techniques, apply them to PIM-DM as a case study, and analyze them quantitatively and qualitatively.

As mentioned earlier, our approaches that are based on search algorithms—namely, the fault-independent test generation (FITG) and the fault-oriented test generation (FOTG)—take as input a processable model of the protocol in the form of a global FSM (GFSM).

Following we will present the GFSM model for the case study protocol (PIM-DM), and use it as an illustrative example to analyze the complexity of the state space and the search problem, as well as illustrate the algorithmic details and principles involved in FITG and FOTG.

5.1 The Protocol Model

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

1. FSM model: Every instance of the protocol, running on a single router, is modeled by a deterministic FSM consisting of: (i) a set of states, (ii) a set of stimuli causing state transitions, and (iii) a state transition function (or table) describing the state transition rules. For a system \( i \), this is represented by the machine \( M_i = (S, \tau_i, \delta_i) \), where \( S \) is a
finite set of state symbols, \( \tau_i \) is the set of stimuli, and \( \delta_i \) is the state transition function \( S \times \tau_i \to S \).

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 \( M_G = (S_G; \tau_G; \delta_G) \), where \( S_G: S_1 \times S_2 \times \cdots \times S_n \) is the global state space, \( \tau_G: \bigcup_{i=1}^{n} \tau_i \) is the set of stimuli, and \( \delta_G \) is the global state transition function \( S_G \times \tau_G \to S_G \).

The fault model is integrated into the GFSM model. For message loss, the transition caused by the message is either nullified or modified, depending on the selective loss pattern. Crashes may be treated as stimuli causing the routers affected by the crash to transit into a \textit{crashed} state \(^1\). Network delays are modeled (when needed) through the delay matrix presented in Chapter 8.

5.2 PIM-DM Model

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

5.2.1 FSM model \( M_i = (S_i, \tau_i, \delta_i) \)

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.

\(^1\)The \textit{crashed} state maybe one of the states already defined for the protocol, like the \textit{empty} state, or may be a new state that was not defined previously for the protocol.
**System States (S)**  Possible states in which a router may exist are:

<table>
<thead>
<tr>
<th>State Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_i$</td>
<td>Router $i$ is a forwarder for the LAN</td>
</tr>
<tr>
<td>$F_i_{\text{Timer}}$</td>
<td>$i$ forwarder with Timer $\tau_{\text{Timer}}$ running</td>
</tr>
<tr>
<td>$NF_i$</td>
<td>Upstream router $i$ a non-forwarder</td>
</tr>
<tr>
<td>$NH_i$</td>
<td>Router $i$ has the LAN as its next-hop</td>
</tr>
<tr>
<td>$NH_i_{\text{Timer}}$</td>
<td>same as $NH_i$ with Timer $\tau_{\text{Timer}}$ running</td>
</tr>
<tr>
<td>$NC_i$</td>
<td>Router $i$ has a negative-cache entry</td>
</tr>
<tr>
<td>$EU_i$</td>
<td>Upstream router $i$ is empty</td>
</tr>
<tr>
<td>$ED_i$</td>
<td>Downstream router $i$ is empty</td>
</tr>
<tr>
<td>$M_i$</td>
<td>Downstream router with attached member</td>
</tr>
<tr>
<td>$NM_i$</td>
<td>Downstream router with no members</td>
</tr>
</tbody>
</table>

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

$$S_i = \begin{cases} 
\{F_i,F_i_{\text{Timer}},NF_i,EU_i\}, & \text{if the router is upstream;} \\
\{NH_i,NH_i_{\text{Timer}},NC_i,M_i,NM_i,ED_i\}, & \text{if the router is downstream.}
\end{cases}$$

**Stimuli (\(\tau\))** 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 \textit{Graft}, in which case the \textit{Rtx} timer is set. Following are the stimuli considered in our study:

1. Transmitting messages: \textit{Graft} transmission ($\text{Graft}_{Tx}$).
2. Receiving messages: \textit{Graft} reception ($\text{Graft}_{Rev}$), \textit{Join} reception ($\text{Join}$), \textit{Prune} reception ($\text{Prune}$), \textit{Graft Acknowledgement} reception ($\text{G Ack}$), \textit{Assert} reception ($\text{Assert}$), and forwarded packets reception ($FPkt$).
3. Timer events: these events occur due to timer expiration ($\text{Exp}$) and include the \textit{Graft} re-transmission timer ($\text{Rtx}$), the event of its expiration ($\text{RtxExp}$), the forwarder-deletion timer ($\text{Del}$), and the event of its expiration ($\text{DelExp}$). We refer to the event of timer expiration as (\textit{Timer Implication}).
4. External host events (\textit{Ext}): include host sending packets ($SPkt$), host joining a group ($HJoin$ or $HJ$), and host leaving a group ($Leave$ or $L$).

$$\tau = \{\text{Join, Prune, Graft}_{Tx}, \text{Graft}_{Rev}, \text{G Ack}, \text{Assert}, FPkt, Rtx, Del, SPkt, HJ, L\}.$$
5.2.2 Global FSM 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, router 2 expecting packets from the LAN, and routers 3 and 4 have negative caches, is given by \( \{F_1, NH_2, NC_3, NC_4\} \).

For the global stimuli \( \tau_G \), subscripts are added to stimuli to denote their originators and recipients (if any). The global transition rules \( \delta_G \) are extended to encompass the router and stimuli subscripts.

5.3 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.

5.3.1 Classification of Transitions

We identify two types of transitions; externally triggered (ET) and internally triggered (IT) transitions. The former is stimulated by events external to the system (e.g., \( H.Join \) or \( Leave \)), whereas the latter is stimulated by events internal to the system (e.g., \( FPkt \) or \( Graft \)).

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

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

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

<table>
<thead>
<tr>
<th>Host Events</th>
<th>ET events</th>
<th>IT events</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPkt</td>
<td>FPkt</td>
<td>Assert, Prune, Join</td>
</tr>
<tr>
<td>H.Join</td>
<td>Graft</td>
<td>GAck, Join</td>
</tr>
<tr>
<td>Leave</td>
<td>Prune</td>
<td></td>
</tr>
</tbody>
</table>

\(^2\) Semantics of the global stimuli and global transitions will be described as needed (see Chapter 7).
5.3.2 Transition Completion

To check for the global system correctness, all stimulated internal transitions should be completed, to bring the system into a stable state. Intermediate (transient) states should not be checked for correctness (since they may 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., \texttt{HJoin} or \texttt{Leave}). Therefore, we should be able to identify a transition based upon its stimuli (either external or internal).

At the end of each complete transition sequence the system exists in either a correct or erroneous stable state. Event-triggered timers (e.g., \texttt{Del}, \texttt{Rtx}) fire at the end of a complete transition.

5.4 Problem Complexity

The problem of finding test scenarios leading the protocol into error (or erroneous states) can be viewed as a search problem in the state space of the protocol (or the system). A conventional reachability analysis approach [LCL87a] would attempt to investigate this space exhaustively, and hence is likely to suffer the 'state space explosion' problem. To circumvent this problem we need to use search reduction techniques using domain-specific information, in this case knowledge of multicast routing.

In this section, we first give the complexity of the state space if explored using exhaustive search approach. Then we discuss the reduction techniques we employ based on notion of equivalence, and discuss the complexity of the state space (as composed of erroneous states and correct states).

5.4.1 Complexity of exhaustive state space search

Exhaustive search attempts to generate and analyze all system states that are reachable from initial system states.

For a system of \(n\) routers where each router may exist in any state \(s_i \in S\), and \(|S| = s\) states, the number of reachable states in the system is bounded by \((s)^n\). To investigate all the transitions, with \(l\) possible transitions, we obtain \(l \cdot (s)^n\) state visits to complete the process. For our case study, \(|S| = 10\). Note that faults (such as message loss and crashes) increase the branching factor \(l\), and may even introduce new states and hence affect \(S\).
In our case the message loss increases the branching factor by the possible selective loss scenarios leading to different global states, while crashes return any state to the empty state, i.e., in our case branching increases by factor of 9.

### 5.4.2 State reduction through equivalence

As explained above, exhaustive search experiences exponential complexity. To circumvent this well-known state space explosion problem, we introduce the notion of equivalence. Intuitively, we take advantage of domain-specific information for multicast routing, where the order in which the states are considered does not matter (e.g., if router $R_1$ or $R_4$ is a forwarder is insignificant, so long as there is only one forwarder). Hence, we can treat the global state as an unordered set of state symbols. We use a symbolic representation as a convenient form of representing the global state to illustrate the notion of equivalence and to help in defining the error and correct states in a succinct manner.

#### 5.4.2.1 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 $G = (q_1^{r_1}, q_2^{r_2}, \ldots, q_m^{r_m})$, where $m = |S|$, $\Sigma r_i = n$. For symbolic representation of topologies where $n$ is unknown $r_i \in [0, 1, 2, 1+, \ast]$ (‘1+’ is 1 or more, and ‘\ast’ is 0 or more).

To satisfy the correctness conditions for PIM-DM, the correct stable global states are those containing no forwarders and no routers expecting packets, or those containing one forwarder and one or more routers expecting packets from the link; symbolically this may be given by: $G_1 = (F^0, NH^0, NC^\ast)$, and $G_2 = (F^1, NH^{1+}, NC^\ast)$. \(^3\)

We use $X$ to denote any state $s_i \in S$. For example, $\{X - F\}^\ast$ denotes 0 or more states $s_i \in S - \{F\}$. This symbolic representation will be used later in this section to estimate the size of the reduced state space.

#### 5.4.2.2 Counting equivalence

Two system states $(q_1, q_2, \ldots, q_n)$ and $(p_1, p_2, \ldots, p_n)$ are strictly equivalent iff $q_i = p_i$, where $q_i, p_i \in S, \forall 1 \leq i \leq n$. However, all routers are assumed inter-changeable and

\(^3\)For convenience, we may represent these two states as $G_1 = (NC^\ast)$, and $G_2 = (F, NH^{1+}, NC^\ast)$. 

45
their behavior 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.

Following from the symbolic representation of \( G \) above, we may represent a state for a system with \( n \) routers as \( \prod_{i=1}^{[S]} s_i^{k_i} \), where \( k_i \) is the number of routers in state \( s_i \in S \) and \( \sum_{i=1}^{[S]} k_i = n \).

**Counting Equivalence:**

Two system states \( \prod_{i=1}^{[S]} s_i^{k_i} \) and \( \prod_{i=1}^{[S]} s_i^{l_i} \) 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. The notion of equivalence (by definition) implies that by investigating the equivalent subspace we can test for protocol correctness. That is, if the equivalent subspace is verified to be correct then the protocol is correct, and if there is an error in the protocol then it must exist in the equivalent subspace.

The notion of counting equivalence also applies to transitions and faults. Those transitions or faults leading to equivalent states are considered equivalent.

### 5.4.2.3 Complexity of the state space with equivalence reduction

Considering counting equivalence, finding the number of equivalent states becomes a problem of combinatorics. The number of equivalent states becomes \( C(n+s-1,n) = \frac{(n+s-1)!}{n!(s-1)!} \), where \( n \) is the number of routers, \( s \) is the number of state symbols, and \( C(x, y) = \frac{x^y}{y!(x-y)!} \), is the number of \( y \)-combination of \( x \)-set [CLR90].

### 5.4.3 Representation of error and correct states

Depending on the correctness definition we may get different counts for the number of correct or error states. To get an idea about the size of the correct or error state space for our case study, we take two definitions of correctness and compute the number of correct states. As shown earlier for the correct states of PIM-DM, we either have: (1) no forwarders with no routers expecting packets from the LAN, or (2) exactly one forwarder with routers expecting packets from the LAN. These conditions we have found to be reasonably sufficient to meet the functional correctness requirements. However, they may not be necessary, hence the search may generate false errors. Proving necessity is part of future work.
The correct space and the erroneous space must be disjoint and they must be complete (i.e., add up to the complete space), otherwise the specification is incorrect. See Appendix B.1 for details.

We present two correctness definitions that are used in our case.

- The first definition considers the forwarder states as $F$ and the routers expecting packets from the LAN as $NH$. Hence, the symbolic representation of the correct states becomes: $(X - NH - F)^*$, or $(NH, F, \{X - F\}^*)$,

and the number of correct states is:

$$C(n + s - 3, n) + C(n + s - 4, n - 2).$$

- The second definition considers the forwarder states as $\{F_i, F_{i-Del}\}$ or simply $F_X$, and the states expecting packets from the LAN as $\{NH_i, NH_{i-Rtx}\}$ or simply $NH_X$. Hence, the symbolic representation of the correct states becomes:

$$(X - NH_X - F_X)^*$$, or $$(NH_X, F_X, \{X - F_X\}^*),$$

and the number of correct states is:
\[ C(n + s - 5, n) + 4 \cdot C(n + s - 5, n - 2) - 2 \cdot C(n + s - 6, n - 3). \]

Refer to Appendix B.2 for more details on deriving the number of correct states.

Figure 5.1 shows the percentage of each of the correct and error state spaces, and how this percentage changes with the number of routers. The figure is shown for the second error definition, but similar results were also obtained for the first definition.

In general, we find that the size of the error state space, according to both definitions, constitutes the major portion of the whole state space. This means that search techniques explicitly exploring the error states are likely to be more complex than others. We take this in consideration when designing our method.
Chapter 6

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 counting equivalence introduced earlier 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.

6.1 Exhaustive Search

The exhaustive search approach is described in this section. Such an approach starts from the initial states, and expands the state space until it is exhausted. This can be implemented using a breadth first or depth first search. The procedure starts from the initial states 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.

To generate all possible initial states, given a set of initial state symbols $I.S.$ and the number of routers $n$, depth first or breadth first search can be used again. For a given number of initial state symbols $|I.S.| = i.s.$, the complexity of this procedure is given by $(i.s.)^n$. For our case study the routers start as either a non-member ($NM$) or empty upstream routers ($EU$), that is $I.S. = \{NM, EU\}$, and $i.s. = 2$.

For details of the above procedures, see Appendix C.1.
6.2 Reduction Using Equivalences

In the above exhaustive search algorithm no reduction techniques were used. We now use the counting equivalence notion to reduce the complexity of the search in three stages of the search:

1. 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 $n = 2$ would be: $\{NM, NM\}, \{NM, EU\}, \{EU, EU\}$.

One procedure that produces such equivalent initial state space given in Appendix C.2. The complexity of the this algorithm is given by $C(n + i.s., - 1, n)$ as was shown in Section 5.4.2. and verified through simulation.

2. The second reduction we use is during state comparison. 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.

3. A 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 $\{NH^1, NC^1, F^1\}$. To achieve this reduction we add flag check before advancing the state machine forward. 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.

6.3 Applying The Method

In this section we discuss how the fault-independent test generation can be applied to the model of PIM-DM. We apply forward search techniques to study correctness of PIM-DM. The study was conducted first without including faults to study the complexity of the algorithms. Then selective message loss was applied and the protocol behavior was studied to analyze the protocol errors.
6.3.1 Method input

The protocol model is provided by the designer or protocol specification, in terms of a transition table or transition rules of the GFSM, and a set of initial state symbols. The design requirements, in terms of correctness in this case, is 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. Furthermore, the detection of equivalence classes needs to be provided by the designer. Currently, we do not automate the detection of equivalent classes. Also, the number of routers in the topology or topologies to be investigated (i.e., on the LAN) has to be specified.

6.3.2 Complexity of forward search for PIM-DM

The procedures presented above were simulated for PIM-DM. The forward search algorithms were simulated for PIM-DM, to study its correctness. This set of results shows behavior of the algorithms without including faults, i.e., when used for verification. 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. A summary of the results is given in Figures 6.1 and 6.2.

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. The error condition is given as in the second error condition in Section 5.4.3. 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.

We notice that there significant reduction in the algorithm complexity with the use of equivalence relations. In particular, 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

\footnote{For our case study, the symmetry inherent in multicast over LANs was used to establish the counting equivalence for states, transitions and faults}
Figure 6.1: Simulation statistics for forward algorithms. Expanded States is the number of visited states and Forwards is the number of forward advances of the state machine.

- Only one error was detected in the two-router case. With the initial state \(\{EU, EU\}\) (i.e., both routers are upstream routers), the system enters the error state \(\{F, NF\}\), where there is a forwarder for the LAN but there are no routers expecting packets or attached members. In this case the Assert process chose one forwarder for the
Figure 6.2: Simulation statistics for forward algorithms. *Transitions* is the number of transient states visited, and *Errors* is the number of stable state errors detected.

LAN, but there were no downstream routers to *Prune* off the extra traffic, and so the protocol causes wasted bandwidth.

- Several errors were detected for the 3-router LAN case:
  - Starting from \{EU, EU, EU\} the system enters the error state \{F, NF, NF\} for a similar reason to that given above.
  - Starting from \{NM, EU, EU\} the system enters the error state \{NC, NF, F\}. By analyzing the trace of events leading to the error we notice that the downstream router *NC* pruned off one of the upstream routers, *NF*, before the *Assert* process takes place to choose a winner for the LAN. Hence the protocol causes wasted bandwidth.
  - Starting from \{NM, EU, EU\} the system enters state \{NH, F, F\}. This is due to the transition table rules, when a forwarader sends a packet, all upstream routers in the *EU* state transit into *F* state. This is not an actual error, however, since the system will recover with the next forwarded packet using *Assert* \(^2\). The detection of this false-error could have been avoided by issuing

\(^2\)This is one case where the correctness conditions for the model are sufficient but not necessary to meet the functional requirements for correctness, thus leading to a false error. Sufficiency and necessity proofs are subjects of future work.
SPkt stimulus before the error check, to see if the system will recover with the next packet sent.

- With message loss, errors were detected for Join and Prune loss. When the system is in \{NH, NH, F\} state and one of the downstream members leaves (i.e., issues L event), a Prune is sent on the LAN. If this Prune is selectively lost by the other downstream router, a Join will not be sent and the system enters state \{NC, NH, NF\}. Similarly, if the Join is lost, the protocol ends up in an error state.

6.4 Challenges and Limitations

In order to generalize the fault-independent test generation method, we need to address several open research issues and challenges. Some of these issues are addressed in later parts of the dissertation, others are to be subjects of future work.

- The topology is an input to the method in terms of number of routers. In Chapter 7 we present a new method that synthesizes the topology automatically as part of the search process. An alternative approach would be to add topology synthesis to FITG. One direction to investigate is to use the symbolic representation presented in Section 5.4, where the use of repetition constructs\(^3\) may be used to represent the LAN topology in general. Ideas similar to those used in [PD96] for cache coherence protocol verification may be investigated, where the state space is split using repetition constructs based on the correctness definition. Note, however, that our problem adds fault modeling, and the state space split will depend on the fault and robustness definition. In addition, correctness varies for different network protocols, so the state space split has to be parametrized as function of correctness. It is important to prove the sufficiency of such approach to generate topology that capture all possible errors.

- Equivalence classes are given as input to the method. Automating identification of equivalence classes is part of future work. In this study we have used symmetries inherent in multicast routing on LANs to utilize equivalence. This symmetry may not exist in other protocols or topologies, hence the forward search may become increasingly complex. Other kinds of equivalence may be investigated to reduce

\(^3\)Repetition constructs include, for example, the \(*\) to represent zero or more states, or the \(1^+\) to represent one or more states, \(2^+\) two or more, so on.
complexity in these cases. Also, other techniques for complexity reduction may be investigated, such as statistical sampling based on randomization or hashing [Hol91]. However, sampling techniques do not achieve full coverage of the state space.

- The topology used in this study is limited to a single-hop LAN. Although we found it quite useful to study multicast routing over LANs, the method needs to be extended to multi-hop LAN to be more general. Chapter 8 introduces the notion of virtual LAN, and future work addresses multi-LAN topologies.

The fault-independent test generation may be used for protocol verification, as was shown, 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. The fault-independent approach, as presented here, is not fit to address performance issues or topology synthesis. These issues are addressed in the coming chapters. The notion of forward search and the use of equivalence for complexity reduction is re-used in our other methods as well.

\footnote{An example of another kind of equivalence is fault dominance, where a system is proven to necessarily reach one error before reaching another, thus the former error dominates the latter error.}
Chapter 7

Fault-oriented Test Generation

In this chapter, we investigate the fault-oriented test generation (FOTG), where the tests are generated for specific faults. In this method, 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. We use the FSM formalism presented in Chapter 5 to represent the protocol. We also re-use some ideas from the other algorithms previously presented, such as forward search and the notion of equivalence for search reduction.

7.1 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 ¹. 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 search techniques similar to those explained earlier in Chapter 6. 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

¹The global state from which FOTG starts is synthesized for a given fault, such as a message to be lost, or a given condition or target event (as will be shown in Chapter 8).
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*.

Much of the algorithmic details are based on *condition → effect* reasoning of the transition rules. This reasoning is emphasized in the semantics of the transition table used in the topology synthesis and the backward search. Section 7.1.1 describes these semantics. In Section 7.2 we describe the algorithmic details of FOTG, and in Section 7.3 we describe how FOTG was applies to PIM-DM in our case study, and present the results and method evaluation. Section 7.4 we discuss the limitations of the method and our findings.

### 7.1 Transition Table

The global state transition may be represented in several ways. Here, we choose a transition table representation that emphasizes the effect of the stimuli on the system, and hence facilitates topology synthesis. The transition table describes, for each stimulus, the conditions of its occurrence. A condition is given as stimulus and state or transition (denoted by *stimulus*.*state/trans*), where the transition is given as *startState → endState*.

We further extend message and router semantics to capture multicast semantics. Following, we present a detailed description of the semantics of the transition table then give the resulting transition table for our case study, to be used later in this chapter.

#### 7.1.1 Semantics of the transition table

In this subsection we describe the message and router semantics, pre-conditions, and post-conditions.

- Stimuli and router semantics: Stimuli are classified based on the routers affected by them. Stimuli types include:

  1. *orig*: stimuli or events occurring within the router originating the stimulus but do not affect other routers, and include HJ, L, *SPkt*, *Graft*$_{Tx}$, *Del* and *Rtx*.
  2. *dst*: messages that are processed by the destination router only, and include *Join*, *G Ack* and *Graft*$_{Rcv}$.
  3. *mcast*: multicast messages that are processed by all other routers, and include *Assert* and *FPkt*.
  4. *mcastDownstream*: multicast messages that are processed by all other downstream routers, but only one upstream router, and includes the *Prune* message.
These types are used by the search algorithm for processing the stimuli and messages. According to these different types of stimuli processing a router may take as subscript \( \text{orig}, \text{dst}, \text{or other} \). \( \text{orig} \) designates the originating router of the stimulus or message, whereas \( \text{dst} \) designates the destination of the message. \( \text{other} \) indicates routers other than the originator. Routers are also classified as \textit{upstream} or \textit{downstream} as presented in Chapter 5.

- **Pre-Conditions:** The pre-conditions in general are of the form \( \text{stimulus.state}/\text{transition} \), where the transition is given as \( \text{startState} \rightarrow \text{endState} \). If there are several pre-conditions, then we can use a logical OR to represent the rule. At least one pre-condition is necessary to trigger the stimulus.

Example of a \( \text{stimulus.state} \) condition is the condition for \textit{Join} message, namely, \( \text{Prune}_{\text{other}} \cdot \text{NH}_{\text{orig}} \), that is, a \textit{Join} is triggered by the reception of a \textit{Prune} from another router, with the originator of the \textit{Join} in \textit{NH}. An example of a \( \text{stimulus.transition} \) condition is the condition for Graft transmission \( HJ(\text{NC} \rightarrow \text{NH}) \); 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 of: (1) \textit{transition}, (2) \textit{condition.transition}, (3) \textit{condition.stimulus}, and (4) \textit{stimulus.transition}.

1. \textit{transition}: has an implicit condition with which it is associated; i.e. ‘\( a \rightarrow b \)’ means ‘if \( a \in GState \) then \( a \rightarrow b \)’. For example, \textit{Join} post-condition \( NF_{\text{dst}} \rightarrow F_{\text{dst}} \), means if \( NF_{\text{dst}} \in GState \) then transition \( NF \rightarrow F \) will occur.

2. \textit{Condition.transition}: is same as (1) except the condition is explicit.

3. \textit{Condition.stimulus}: if the condition is satisfied then the stimulus is triggered. For example, \textit{Prune} post-condition ‘\( \text{NH}_{\text{other}} \cdot \text{Join}_{\text{other}} \)’, means that for all \( \text{NH}_x \in GState \) (where \( x \) is not equal to \textit{orig}) then have router \( x \) trigger a \textit{Join}.

4. \textit{Stimulus.transition}: has the transition condition implied as in (1) above. For example, \textit{Graft}_{\text{rev}} post-condition ‘\( \text{GAck}(NF_{\text{dst}} \rightarrow F_{\text{dst}}) \)’, means if \( NF_{\text{dst}} \in GState \), then the transition occurs and \textit{GAck} is triggered.

\[ ^2 \text{Network faults, such as message loss, may cause the stimulus not to take effect. For example, losing a Join message will cause the event of Join reception not to take effect.} \]

\[ ^3 \text{This does not appear in our case study.} \]
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 \( F_{\text{dst}_\text{Del}} \rightarrow F_{\text{dst}}, NF_{\text{dst}} \rightarrow F_{\text{dst}} \), which means \((F_{\text{dst}_\text{Del}} \rightarrow F_{\text{dst}}) \) XOR \((NF_{\text{dst}} \rightarrow F_{\text{dst}})\). \(^4\)

However, Prune post-conditions are \( F_{\text{dst}} \rightarrow F_{\text{dst}_\text{Del}}, NH_{\text{other}} \cdot \text{Join}_{\text{other}} \), which implies that the transition will occur if \( F_{\text{dst}} \in G\text{State} \) AND a \text{Join} will be triggered if \( NH \in G\text{State} \).

Following is the transition table used in our case study.

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>Pre-conditions</th>
<th>Post-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join</td>
<td>Prune_{other}, NH_{orig}</td>
<td>( F_{\text{det}<em>\text{Del}} \rightarrow F</em>{\text{det}}, NF_{\text{det}} \rightarrow F_{\text{det}} )</td>
</tr>
<tr>
<td>Prune</td>
<td>L, NC, FPkt, NC</td>
<td>( F_{\text{det}} \rightarrow F_{\text{det}<em>\text{Del}}, NH</em>{\text{other}} \cdot \text{Join}_{\text{other}} )</td>
</tr>
<tr>
<td>Graft_{Tx}</td>
<td>HJ {NC \rightarrow NH}, RtExp{NH_{\text{Retx}} \rightarrow NH}</td>
<td>( \text{Graft}<em>{\text{Rec}} {NH \rightarrow NH</em>{\text{Retx}}} )</td>
</tr>
<tr>
<td>Graft_{Rec}</td>
<td>( \text{Graft}<em>{\text{Rec}} {NH \rightarrow NH</em>{\text{Retx}}} )</td>
<td>( \text{Graft}<em>{\text{Rec}} {NH \rightarrow NH</em>{\text{Retx}}} )</td>
</tr>
<tr>
<td>Ack</td>
<td>( \text{Graft}_{\text{Rec}} \cdot F )</td>
<td>( NH_{\text{Retx}} \cdot F \rightarrow NH_{\text{Retx}} )</td>
</tr>
<tr>
<td>Assert</td>
<td>( \text{FPkt}<em>{\text{other}} \cdot F</em>{\text{orig}} )</td>
<td>( F_{\text{other}} \rightarrow NF_{\text{other}} )</td>
</tr>
<tr>
<td>FPkt</td>
<td>( \text{Spkt} \cdot F )</td>
<td>( \text{Prune}. {NM \rightarrow NC}, ED \rightarrow NH, M \rightarrow NH, EU_{\text{other}} \rightarrow F_{\text{other}}, F_{\text{other}} \cdot \text{Assert} )</td>
</tr>
<tr>
<td>Rtx</td>
<td>( \text{RtxExp} )</td>
<td>( \text{Graft}<em>{\text{Rec}} {NH</em>{\text{orig}} \rightarrow NH_{\text{orig}}} )</td>
</tr>
<tr>
<td>Del</td>
<td>( \text{DelExp} )</td>
<td>( F_{\text{orig}<em>\text{Del}} \rightarrow NF</em>{\text{orig}} )</td>
</tr>
<tr>
<td>SPkt</td>
<td>Ext</td>
<td>( \text{FPkt}{EU_{\text{orig}} \rightarrow F_{\text{orig}}} )</td>
</tr>
<tr>
<td>H Join</td>
<td>Ext</td>
<td>( NM \rightarrow M, \text{Graft}_{\text{Rec}} {NC \rightarrow NH} )</td>
</tr>
<tr>
<td>Leave</td>
<td>Ext</td>
<td>( M \rightarrow NM, \text{Prune}. {NH \rightarrow NC}, \text{Prune}. {NH_{\text{Retx}} \rightarrow NC} )</td>
</tr>
</tbody>
</table>

The above pre-conditions can be derived automatically from the post-conditions. The ‘PreConditions’ procedure takes as input one form of the conventional transition table and produces the pre-condition semantics. See Appendix D for details of such procedure.

### 7.1.1.2 State Dependency Table

To aid in test sequence synthesis through the backward implication procedure, we construct what we call a state dependency table. This table can be inferred automatically from the transition table. We use this table to improve the performance of the algorithm and for illustration.

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

\(^4\)There is an implicit condition that can never be satisfied in both statements, which is the existence of \( \text{dst} \) in only one state at a time.
The 'dependencyTable' procedure in Appendix D generates the dependency table from the transition table of conditions. For \( s \in I.S. \) a symbol denoting initial state is added to the array entry. For our case study \( I.S. = \{NM, EU\} \). Based on the above transition table, following is the resulting state dependency table: \(^5\)

<table>
<thead>
<tr>
<th>State</th>
<th>Possible Backward Implication(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_i )</td>
<td>( F_{i\rightarrow del} ) ( \rightarrow ) ( EU_i ), ( F_{i\rightarrow del} ) ( \rightarrow ) ( NF_i ), ( \rightarrow ) ( F_{i\rightarrow del} ) ( \rightarrow ) ( NF_i ), ( \rightarrow ) ( EU_i )</td>
</tr>
<tr>
<td>( F_i\rightarrow del )</td>
<td>( Prune ) ( \leftarrow ) ( F_i )</td>
</tr>
<tr>
<td>( NF_i )</td>
<td>( Del ) ( \leftarrow ) ( F_{i\rightarrow del} ), ( \rightarrow ) ( F_i )</td>
</tr>
<tr>
<td>( NH_i )</td>
<td>( Prune ) ( \leftarrow ) ( NH_i ), ( \rightarrow ) ( NF_i ), ( \rightarrow ) ( F_{i\rightarrow del} ) ( \rightarrow ) ( NF_i ), ( \rightarrow ) ( ED_i )</td>
</tr>
<tr>
<td>( NH_i\rightarrow RTS )</td>
<td>( Prune ) ( \leftarrow ) ( NH_i ), ( \rightarrow ) ( NH_i )</td>
</tr>
<tr>
<td>( NC_i )</td>
<td>( Prune ) ( \leftarrow ) ( NM_i ), ( \rightarrow ) ( NH_i ), ( \rightarrow ) ( NH_i )</td>
</tr>
<tr>
<td>( EU_i )</td>
<td>( \leftarrow I.S. )</td>
</tr>
<tr>
<td>( ED_i )</td>
<td>( \leftarrow I.S. )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( \rightarrow I.S. )</td>
</tr>
<tr>
<td>( NM_i )</td>
<td>( \leftarrow I.S. )</td>
</tr>
</tbody>
</table>

In cases where the stimulus affects more than one router (e.g., multicast Prune), multiple states need to be simultaneously implied in one backward step, otherwise an \( I.S. \) may not be reached. To do this, the transitions in the post-conditions of the stimulus are traversed, and any states in the global state that are endStates are replaced by their corresponding startStates. For example, \( \{M_i, NM_j, F_k\} \rightarrow \{NH_i, NC_j, F_k\} \). This is taken care of by the backward implication section described later.

### 7.2 FOTG details

As previously mentioned, our FOTG approach consists of three phases: I) synthesis of the global state to inspect, II) forward implication, and III) backward implication. These phases are explained in more detail in this section. In Section 7.3 we present an illustrative example for the these phases.

#### 7.2.1 Synthesizing the Global State

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

\(^5\)The possible backward implications are separated by ‘commas’ indicating ‘OR’ relation.
1. The global state is initially empty and the inspected stimulus is initially set to the stimulus investigated.

2. For the inspected stimulus, the state(s) (or the `startState`(s) of the transition) of the post-condition are obtained from the transition table. If these states do not exist in the global state, and cannot be inferred therefrom, then they are added to the global state.

3. For the inspected stimulus, the state(s) (or the `endState`(s) of the transition) of the pre-condition are obtained. If these states do not exist in the global state, and cannot be inferred therefrom, then they are added to the global state.

4. Get the stimulus of the pre-condition of the inspected stimulus, call it `newStimulus`. If `newStimulus` is not external `(Ext)` , then set the inspected stimulus to the `newStimulus`, and go back to step 2.

The second step considers post-conditions and adds system components that will be affected by the stimulus. While the third and fourth steps synthesize the components necessary to trigger the stimulus. The procedure given in Appendix D synthesizes minimum topologies necessary to trigger a given stimulus of the protocol. 

Note that there may be several pre-conditions or post-conditions for a stimulus, in which case several choices can be made. These represent branching points in the search space. At the end of this stage, the global state to be investigated is obtained.

### 7.2.2 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, as will be shown in Section 7.3. Forward implication uses the forward search techniques described earlier in Chapter 6.

According to the transition completion concept (see Section 5.3.2), 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. This will be implied in the forward implication algorithm to simplify the discussion.

7.2.3 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 $I.S.$

The state dependency table described in Section 7.1.1.2 is used in the backward search.

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. A set of visited states is maintained to avoid looping. 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. For $orig$ and $dst$ only the originator and destination of the stimulus is rolled back, respectively. For $mcast$, all affected states are rolled back except the originator. $mcastDownstream$ is similar to $mcast$ except that all downstream routers or states are rolled back, while only one upstream router (the destination) is rolled back. Appendix D shows procedures ‘Backward’ and ‘Rewind’ that implement the above steps.

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 of this contradiction 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, for instance, \textit{Join} stimulus with global state \{\textit{NH}, \textit{NF}\}, or \textit{Assert} stimulus with global state \{\textit{F}, \textit{NH}, \textit{NC}\}.

- Failure of consistency check: to maintain consistency of the transition rules in the reverse direction, we must check that every backward step has an equivalent forward step. To achieve this, 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). An example of this inconsistency exists when the stimulus is \textit{FPkt} and \(GState = \{\textit{F}, \textit{NF}, \textit{EU}\}\), where \(\textit{EU} \rightarrow \textit{F}\) is a post condition for \textit{FPkt}. See Appendix D for the consistency check procedure.

### 7.3 Applying The Method

In this section we discuss how the fault-oriented test generation can be applied to the model of PIM-DM. Specifically, we discuss in details the application of FOTG to the robustness analysis of PIM-DM in the presence of single message loss and machine crashes. We first walk through a simple illustrative example. Then we present the results of the case study in terms of correctness violations captured by the method.

#### 7.3.1 Method input

The protocol model is provided by the designer or protocol specification, in terms of a transition table\(^6\), and the semantics of the messages. In addition, a list of faults to be studied is given as input to the method. For example, definition of the fault as single selective protocol message loss, applied to the list of messages \{\textit{Join}, \textit{Prune}, \textit{Assert}, \textit{Graft}\}. Also a set of initial state symbols, in our case \{\textit{NM}, \textit{EU}\}. A definition of the design requirement, in this case definition of correctness, is also provided by the specification. The rest of the process is automated.

#### 7.3.2 Illustrative example

Figure 7.1 shows the phases of FOTG for a simple example of a \textit{Join} loss. Following are the steps taken for that example:

\(^6\)The traditional input/output transition table is sufficient for our method. The pre/post-condition transition table can be derived automatically therefrom.
### Synthesizing the Global State

1. **Join**: startState of the post-condition is $NF_i \implies G_I = \{NF_k\}$
2. **Join**: state of the pre-condition is $NH_i \implies G_I = \{NH_i, NF_k\}$, goto Prune
3. **Prune**: startState of the post-condition is $F_k$ which can be implied from $NF_k$ in $G_I$
4. **Prune**: state of the pre-condition is $NC_j \implies G_I = \{NH_i, NF_k, NC_j\}$, goto $L$ (External event)
5. The startState of the post-condition is $NH$ which can be implied from $NC$ in $G_I$

#### Forward implication

- without loss: $G_I = \{NH_i, NF_k, NC_j\} \xrightarrow{\text{Join}} G_{I+1} = \{NH_i, F_k, NC_j\}$
- loss w.r.t. $R_i$: $\{NH_i, NF_k, NC_j\} \rightarrow G_{I+1} = \{NH_i, NF_k, NC_j\}$ error

#### Backward implication

- $G_I = \{NH_i, NF_k, NC_j\} \xrightarrow{\text{Prune}} G_{I-1} = \{NH_i, F_k, NC_j\} \xrightarrow{\text{FPkt}} G_{I-2} = \{M_i, F_k, NM_j\}$
- $G_{I-3} = \{M_i, EU_k, NM_j\} \xrightarrow{\text{SPkt}} G_{I-4} = \{NM_k, EU_k, NM_j\} = I.S.$

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

---

**Figure 7.1**: Join topology synthesis, forward/backward implication
7.3.3 Summary of Results

In this section we briefly discuss the results of applying our method to PIM-DM. The analysis is conducted for single message loss and momentary loss of state. For a detailed analysis of the results see Appendix D.6.

7.3.3.1 Single message loss

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

Our method as presented here, however, may not be generalized to transform any type of timing problem into sequencing problem. This topic bears more research in the future.

Join: A scenario similar to that presented in Section 7.3.2 incurred an error. In this case, the robustness violation was not allowing another chance to the downstream router to send a Join. A suggested fix would be to send another prune by $F_{Del}$ before the timer expires.

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

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

Graft: A Graft message is acknowledged by GAck, hence the protocol did not incur error when the Graft message was lost with non-interleaved external events. The protocol is robust to Graft loss with the use of Rtx timer. Adversary external conditions are interleaved during the transient states and the Rtx timer is cleared, such that the adverse event will not be overridden by the Rtx mechanism.
To clear the *Rtx* timer, a transition should be created from *NH*<sub>Rrx</sub> to *NH* which is triggered by a *GAck* according to the state dependency table (*NH*\(\xrightarrow{G Ack}\) *NH*<sub>Rrx</sub>). This transition is then inserted in the event sequence, and forward and backward implications are used to obtain the overall sequence of events illustrated in figure 7.2. In the first and second scenarios (I and II) no error occurs. In the third scenario (III) when a *Graft* followed by a *Prune* is interleaved with the *Graft* loss, the *Rtx* timer is reset with the receipt of the *Gack* for the first *Graft*, and the systems ends up in an error state. A suggested fix is to add sequence numbers to *Grafts*, at the expense of added complexity.

### 7.3.3.2 Loss of State

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

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>Pre-cond</th>
<th>Post-cond (stimulus.state/trans)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Crash</em></td>
<td>Ext</td>
<td>{NM,M,NH,NC,NH&lt;sub&gt;Rrx&lt;/sub&gt;} → ED, {F,F&lt;sub&gt;Del&lt;/sub&gt;,NF} → EU</td>
</tr>
</tbody>
</table>

The FSM resumes function immediately after the crash (i.e. further transitions are not affected). We analyze the behavior when the crash occurs in any router state. For every state, a topology is synthesized that is necessary to create that state. We leverage the topologies previously synthesized for the messages. For example, state *F<sub>Del</sub>* may
be created from state $F$ by receiving a $Prune$ ($F_{Del} \xrightarrow{Prune} F$). Hence we may use the topologies constructed for $Prune$ loss to analyze a crash for $F_{Del}$ state.

Forward implication is then applied, and behavior after the crash is checked for correct packet delivery. To achieve this, host stimuli (i.e. $SPkt$, $HJ$ and $L$) are applied, then the system state is checked for correctness.

In lots of the cases studied, the system recovered from the crash (i.e. the system state was eventually correct). The recovery is mainly due to the nature of PIM-DM; where protocol states are re-created with reception of data packets. This result is not likely to extend to protocols of other natures; e.g. PIM Sparse-Mode [EFH+97].

However, in violation with robustness requirements, there existed cases in which the system did not recover. In figure 7.3, the host joining in (II, a) did not have the sufficient state to send a $Graft$ and hence gets join latency until the negative cache state times out upstream and packets are forwarded onto the LAN as in (II, b).

![Figure 7.3: Crash leading to join latency](image)

In figure 7.4 (II, a), the downstream router incurs join latency due to the crash of the upstream router. The state is not corrected until the periodic broadcast takes place, and packets are forwarded onto the LAN as in (II, b).

### 7.4 Challenges and Limitations

Although we have been able to apply FOTG to PIM-DM successfully, there remains to be some challenges and open issues that we discuss in this section. Some of these challenges are addressed in later chapters to generalize FOTG to apply to a wider range of protocols and evaluations. Others are still under study and will be addressed in the future work section. We also discuss our experience, findings and insights that we have developed through the process of designing and implementing 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 in a multi-hop topology; even if each LAN in the topology operates correctly, the inter-LAN interaction may introduce erroneous behaviors. Applying FOTG to multi-LAN topologies is part of future research.

The analysis for our case studies was done for network layer protocols, namely multicast routing in a single-hop LAN environment. We did not consider network delays. In order to study end-to-end protocols network delays must be considered in the model. In Chapter 8 we extend the notion of the LAN to include end-to-end delay semantics, we call it a virtual LAN.

The evaluation criteria for protocols, especially for end-to-end protocols, usually emphasize performance, in addition to correctness and robustness. Semantics of performance measures, and incorporating richer timing semantics (that are often part of these measures) are discussed in Chapter 8.

Minimal topologies that are necessary and sufficient to trigger the stimuli, may not be sufficient to capture all correctness violations. For example, in some cases it may require one member to trigger a Join, but two members to experience an error caused by Join loss. Hence, the topology synthesis stage must be complete in order to capture all possible errors. To achieve this we propose to use the symbolic representation. For example, to cover all topologies with one or more members we use \( (M^{1+}) \). Integration of this notation with the full method is part of future work.
• The efficiency of the backward search may be increased using reduction techniques, such as equivalence of states and transitions (similar to the ones presented in Chapter 6). This is still a topic of research and is part of future work. Possible directions to investigate include using heuristics to direct the search and avoid backtracking. For example, a heuristic function may be used to give weights to edges that are more likely to succeed or give shorter test sequences. Such function may be based on statistical sampling to develop a profile for the protocol behavior, or may utilize insights about the protocol, such as the transition completion tables described in Chapter 5.

Instead of performing complete backward search until initial state is reached or unreachability is detected, the algorithm may use information about reachable states to reduce the search. This information about state reachability could be obtained simply by storing previous sequences and states visited. Alternatively, the designer may provide information—based on protocol-specific knowledge—about reachable states, through a compact representation thereof.

• 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. 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 (as presented here) may be used to evaluate behavior of specified mechanisms in the presence of network failures, but is not a general protocol verification tool. If used for verification, FOTG would start from the error states and search backwards. In our case study, however, we have noticed that the error state space constitutes the majority of the state space for large number of routers, hence the complexity grows. See Chapter 5.

• 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 non-reachable states, until they are detected as unreachable in the last backward search phase. Adding reachability detection in the early stages of FOTG is subject of future work. However, statistics collected in our case study (see Appendix D) 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.
- The error in the *Graft* mechanism was only detected after inserting an adversary interleaved event. The logic presented in this document deals with the timing mechanisms in our case study. In order to generalize a mechanism to convert timing problems into sequencing problems (if possible), further study must be conducted. Future work in this area should also include classification of timing problems.

Another type of timing problems is addressed in Chapter 8 using the delay matrix, virtual LAN, and integration of timing semantic into the model and the search algorithm.

We believe that the strength of our fault-oriented method, as was demonstrated, lies in its ability to construct the necessary conditions for erroneous behavior by starting directly from the fault, and avoiding the exhaustive walk of the state space. Also, converting timing problems into sequencing problems (as was shown for *Graft* analysis) reduces the complexity required to study timers. FOTG as presented in this chapter seems best fit to study protocol robustness in the presence of faults. Faults presented in our studies include single selective loss of protocol messages and router crashes.
Chapter 8

Performance Evaluation of End-to-End Multipoint Protocols

In this chapter we extend the fault-oriented test generation method to study performance of end-to-end multipoint mechanisms. We introduce the concept of a virtual LAN to represent the underlying network, integrate timing and delay semantics into our model and use performance criteria to drive our synthesis algorithm.

As a case study, we identify the timer suppression mechanism as a building block used by several multipoint protocols and analyze its worst and best case performance behaviors in a systematic fashion.

8.1 Timer Suppression in Multipoint Protocols

As described in section 2.1.2, the timer suppression mechanism is a common technique used to alleviate the Ack-implosion problem. It is employed in several multipoint protocols.

In this mechanism, a member of a multicast group that has detected the loss of a data packet sends a multicast request for recovery. Other members of the group that have previously received the data packet schedule the transmission of a response. In general, randomized timers are used in scheduling the response. While a response timer is running at one end-system (host), if a response is received from another end-system then the response timer is suppressed to reduce the number of responses per request. Consequently, the response time may be delayed to allow for more suppression.

The two main performance evaluation criteria used in this case are the overhead of response messages and the time to recover from packet loss. According to the relative delays between the group members and the timer settings, the mechanism exhibits different performance. In this chapter, our method attempts to obtain scenarios of best case and worst case performance according to the above evaluation criteria.
Here, we describe the role of the timer suppression mechanism in some of these protocols:

- IP-multicast protocols, such as PIM [EFH+97] and IGMP [Fen97], use the timer suppression mechanism on LANs to reduce the number of control messages sent in the Join/Prune and Assert mechanisms.

- For reliable multicast schemes, such as scalable reliable multicast (SRM) [FJL+96], the mechanism is used to alleviate the ack-implosion problem, or reduce number of responses on a LAN as in multicast ftp (MFTP [MRTW98]). Variants of the SRM timers are used in registry replication (e.g., RRM [GYE98]) and adaptive web caching [ZMN+98].

- In multicast address allocation schemes, such as the address allocation protocol (AAP [Han98]) and session directory (sdr) [Han96b], the timer suppression mechanism is used in the request-response protocol to avoid an implosion of responses during the collision detection phase.

- In the context of active services [AMK98], e.g., in a service offered by media gateway servers, multicast damping is used to launch one service agent ‘servent’ from a pool of servers.

There are other applications of the timer suppression mechanism in areas of self-organizing hierarchies (SCAN [GAE19]), and transport protocols (e.g., XTP [ACFS96] and RTP [SCFJ96]).

8.2 The model

The model is a processable representation of the system under study, that enables automation of our method. The overall model consists of three parts: A) the protocol, B) the topology, and C) the faults.

The protocol model is based on the global finite state machine (GFSM) model presented in chapter 5. Instead of having each finite state machine be a router, here it is an end-system (host) running an instance of the multipoint protocol.

8.2.1 The Topology Model

The topology cannot be captured simply by one metric. Indeed, its dynamics may be too complex to model and sometimes intractable. We capture two primary characteristics of
the topology; the delays and loss patterns (see the fault model). We use a virtual LAN (VLAN) model to represent the underlying network topology and multicast distribution tree. The VLAN captures delay semantics using a delay matrix D (see Figure 8.1), where $d_{i,j}$ is the delay from system $i$ to system $j$.

![Figure 8.1: The virtual LAN and the delay matrix](image)

**8.2.2 The Fault Model**

The general fault model was defined in chapter 3. Here, we only consider packet loss and extended delays between end systems. Selective loss is the general form of packet loss that may be experienced by a multipoint application; where a multicast message may be received by some systems but not others. The loss of a message, by a specific system, prevents its reception, and hence prevents the transition that it was meant to trigger at that system.

**8.3 Applying The Method**

To apply the method, the designer specifies the protocol model to be evaluated, and the criteria of evaluation. In this paper we address performance criteria. The algorithm operates on the specified model and obtains a set of constraints (or relations) between delays and timers, to stress the protocol according to the evaluation criteria. The stress scenarios are then obtained by satisfying these constraints by assigning topology delays or timer values.
In this section, we give an outline of the algorithmic details of the method. Then we describe two tasks relating to the timer suppression mechanism in general, to which our method may be applied.

8.3.1 Algorithm Outline

This section outlines the algorithm used for test synthesis. This algorithm is a variant of the fault-oriented test generation (FOTG) algorithm presented in 7. It includes the topology synthesis, the backward search and the forward search stages. Here, we only describe those aspects of our algorithm that deal with timing and performance semantics. The basic algorithm passes through three main steps (1) the target event identification, (2) the search, and (3) the task specific solution.

1. **The target event**: The algorithm used in our method starts from a given event, called the ‘target event’. The target event (e.g., sending a message) is identified by the designer as one relevant to the protocol evaluation criteria (e.g., the overhead of a given message).

2. **The search**: Three steps are taken in the search: (a) identifying conditions, (b) obtaining sequences, and (c) formulating inequalities.

   (a) **Identifying conditions**: The algorithm uses the transition rules to identify conditions and transitions necessary to trigger the target event or prevent it, these are called *wanted transitions* and *unwanted transitions*, respectively.

   (b) **Obtaining sequences**: Once the previous transitions are identified, the algorithm uses i) backward search, and ii) forward search, to build sequences of events leading to these transitions and calculates the times of these events as follows.

   i. **Backward search** is used to identify events preceding the wanted and unwanted transitions, and uses implication rules that operate on the protocol’s transition table. Some implication rules include timing semantics to account for network delays or durations of timers. Implication rules include: 1) reception/transmission pairing, where reception of a transmitted message is implied after applying the network delay if the message is not lost, 2) timer expiration, where the event of firing the timer is implied after the expiration period, and 3) state creation, where previous states are implied by reversing the state transition rules.
ii. **Forward search** is used to verify the backward steps taken and to eliminate contradictions. Every backward step must correspond to a valid forward step, otherwise backtracking is used to recover from contradictions.

(c) **Formulating inequalities:** Based on the conditions and transitions previously obtained, along with the timed sequences leading to the transitions, the algorithm formulates timing constraints (in the form of inequalities, in terms of delays and timer settings) that would trigger the wanted transitions and avoid the unwanted transitions.

3. **Task specific solution:** The output of the search includes a set of event sequences, and constraints (in the form of timing inequality relations between timers and network delays) that satisfy the given condition or evaluation criterion.

The output is then processed to find a solution depending on the task definition, whether it is topology synthesis or timer configuration.

These steps are further discussed in section 8.5 and illustrated by a case study.

### 8.3.2 Task Definition

We identify two kinds of tasks to which our method may be applied: 1) topology synthesis, and 2) timer configuration.

1. **Topology synthesis** is performed when the timer values are known, and the objective is to identify the topology (i.e., $D$ matrix) that produces the best or worst case behavior.

2. **Timer configuration** is performed when the topology is given (i.e., the $D$ matrix is known), and the timer values are being determined (i.e., variables). The solution in this case is the timer expiration values or ranges that cause best or worst case behavior.

### 8.4 Case Study: The Timer Suppression Mechanism

As explained earlier in chapter 2.1, the timer suppression mechanism is used in various multipoint protocols. We believe it is a good building block to analyze as a first end-to-end case study, since it is rich in multicast and timing semantics, and can be evaluated
using standard performance criteria. In this section, we present a simple description of
the mechanism, then present its model, used thereafter in the analysis.

The timer suppression mechanism involves a request \( q \) and one or more responses \( p \).
When a system \( Q \) detects the loss of a data packet it sets a request timer and multicasts a
request \( q \). When a system \( i \) receives \( q \) it sets a response timer (randomly or as a function
of some parameter), the expiration of which, after duration \( \text{Exp}_i \), triggers a response, \( p \).
If the system \( i \) receives a response \( p \) from another system \( j \) before its timer expires, it
suppresses its own response.

### 8.4.1 Evaluation Criteria

Two criteria may be used to evaluate the performance of this mechanism:

1. The first is the number of response messages per request. In this case, we define the
   worst case behavior to be one that produces the maximum number of responses per
   request. As an extreme case, this occurs when no suppression takes place; i.e., all
   potential responders that set their timers do indeed respond.

2. The second performance criterion is the response delay, i.e., the time taken by the
   requester to receive a response. The worst scenario in this case is one that experiences
   maximum response delay.

### 8.4.2 Timer Suppression Model

Following is the model of the timer suppression mechanism we use in the rest of the paper.

### 8.4.2.1 Protocol states (\( S \))

Following is the state symbol table for our model of the timer suppression mechanism,
along with the meaning of each symbol.

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D )</td>
<td>potential responder</td>
</tr>
<tr>
<td>( D_T )</td>
<td>responder with the response timer set</td>
</tr>
<tr>
<td>( R )</td>
<td>original state of the requester</td>
</tr>
<tr>
<td>( R_T )</td>
<td>requester with the request timer set</td>
</tr>
</tbody>
</table>
8.4.2.2 Stimuli or Events ($\tau$)

1. Sending/receiving messages: sending response ($p_t$) and request ($q_t$), receiving response ($p_r$) and request ($q_r$).

2. Timer and other events: the events of firing the request timer $Req$ and response timer $Res$. $L$ denotes detecting packet loss.

8.4.2.3 Notation

Following is a description of the notations used in the transition table.

- An event subscript denotes the system initiating the event, e.g., $p_{t_i}$ is response sent by system $i$, while the subscript $m$ denotes multicast reception, e.g., $p_{r_m}$ denotes receipt of a response by all members of the group if no loss occurs. When system $i$ receives a message sent by system $j$, this is denoted by the subscript $i,j$, e.g., $p_{r_{i,j}}$ is system $i$ receiving response from system $j$.

- The state subscript $T$ is used to denote the existence of a timer, and is used by the algorithm to apply the ‘timer implication’ to fire the timer event after the expiration period.

- A state transition has a start state and an end state and is expressed in the form $startState \rightarrow endState$ (e.g., $D \rightarrow DT$). It implies the existence of a system in the $startState$ (i.e., $D$) as a condition for the completion of the transition to the $endState$ (i.e., $DT$).

- An effect in the transition table may contain state transition and stimulus in the form $(startState \rightarrow endState).stimulus$, which indicates the triggering of the stimulus if the state transition occurs. An effect may contain several transitions (e.g., ‘Trans1, Trans2’), which indicates that out of these transitions all transitions with satisfied conditions will occur.

8.4.2.4 Transition Table ($\delta$)

Following is the transition table for the timer suppression mechanism.
The model contains one requester \( Q \) and several potential responders (e.g., \( i \) and \( j \)).

Let \( t_0 \) be the time at which \( Q \) sends the request \( q \). All the potential responders initially exist in state \( D \). The request sent by \( Q \) is received by \( i \) and \( j \) at times \( d_{Q,i} \) and \( d_{Q,j} \), respectively. When the request \( q \) is sent, the requester transitions into state \( R_T \) by setting the request timer. Upon receiving a request, a potential responder in state \( D \) transitions into state \( D_T \), by setting the response timer. The time at which an event occurs is given by \( t(event) \), e.g., \( qr_j \) occurs at \( t(qr_j) \).

### 8.4.2.5 Implication Rules

The backward search uses the following cause-effect implication rules:

1. **Transmission/Reception** (\( \text{Tx.Rcv} \)): By the reception of a message, the algorithm implies the transmission of that message — without loss — sometime in the past (after applying the network delays). An example of this implication is \( p_{r_{i,j}} \Leftarrow p_t \), where \( t(p_{r_{i,j}}) = t(p_t) + d_{j,i} \).

2. **Timer Expiration** (\( \text{Tmr.Exp} \)): When a timer expires, the algorithm infers that it was set \( Exp \) time units in the past, and that no event occurred during that period to reset the timer. An example of this implication is \( Res_i(D_i \Leftarrow D_{T_i}) \Leftarrow D_{T_i} \), where \( t(Res_i) = t(D_{T_i}) + Exp_i \), and \( Exp_i \) is the duration of the response timer \( Res_i \).

3. **State Creation** (\( \text{St.Cr} \)): A state is created from another by reversing the transition rules and going towards the \textit{startState} of the transition. For example, \( D_{T_i} \Leftarrow (D_{T_i} \Leftarrow D_i) \).

---

1. Since there is only one requester, we simply use \( q_t \) instead of \( q_{t_Q} \), and \( q_{r,i} \) instead of \( q_{r_{Q,i}} \).
2. The time of a state is when the state was first created, so \( t(D_{T_i}) \) is the time at which \( i \) transited into state \( D_T \).
3. We use the notation \( \text{Event.Effect} \) to represent a transition.
In the following sections we use the above model and algorithm to synthesize worst and best case behavior scenarios according to the protocol overhead and the response time performance criteria.

8.5 Protocol Overhead Analysis

In this section, we conduct worst and best case performance analyses for the timer suppression mechanism with respect to the number of responses triggered per request. Initially, we assume no loss of messages until recovery, and we assume that the request timer is high enough that the recovery will occur within one request round. Multiple request rounds that involve multiple timers is discussed in Appendix E.3.

8.5.1 Worst-case analysis

Worst-case overhead analysis in our study aims to attain scenarios that experience the maximum number of responses per request. In this section we present our algorithm to obtain the inequalities that lead to worst-case scenarios. These inequalities are given in terms of network delays and timer expiration values.

8.5.1.1 Target event and conditions

Since the overhead in this case is measured as the number of response messages, the designer identifies the event of triggering a response $p_t$ as the target event, and the goal is to maximize the number of response messages.

8.5.1.2 The search

As previously described in section 8.3.1, there are three main steps for the search algorithm:

1. identifying the target’s wanted and unwanted transitions,
2. obtaining sequences leading to the wanted and unwanted transitions, and calculating the times for these sequences, and
3. formulating the inequalities that achieve the time constraints required to invoke wanted transitions and avoid unwanted transitions.

- Identifying conditions:
The algorithm searches for the transitions necessary to trigger the target event, and their conditions, recursively. These are called *wanted transitions* and *wanted conditions*, respectively. In addition, the algorithm searches for transitions that nullify the target event or invalidate any of its conditions. These are called *unwanted transitions*.

In our case the target event is the transmission of a response (i.e., $p_t$). The algorithm identifies transition $res_{tmr} [Res.(D_T \rightarrow D).p_t]$ as a *wanted transition* and its condition $D_T$ as a *wanted condition*. Transition $rcv_{req} [q_r.D \rightarrow D_T]$ is also identified as a *wanted transition* since it is necessary to create $D_T$.

The *unwanted transition* is identified as transition $rcv_{res} [p_r.D_T \rightarrow D]$ since it alters the $D_T$ state without invoking $p_t$.

- **Obtaining sequences:**

  Using backward search, the algorithm obtains sequences and calculates time values for the following transitions: (1) the wanted transition, $res_{tmr}$, (2) the wanted transition $rcv_{req}$, and (3) the unwanted transition $rcv_{res}$, as follows:

1. To obtain the sequence of events for transition $res_{tmr}$, the algorithm applies implication rules $Tmr_{Exp}$, $St_{Cr}$, $Tx_{Rcv}$ in that order, and we get

   \[ Res_i.(D_i \leftarrow D_{Ti}).p_t_i \leftarrow q_{ri}.(D_{Ti} \leftarrow D_i) \leftarrow q_{ri} \cdot \]

   Hence the calculated time for $t(p_t_i)$ becomes

   \[ t(p_t_i) = t_0 + d_{Q,i} + E_{pi} \cdot \]

2. To obtain the sequence of events for transition $rcv_{req}$ the algorithm applies implication rule $Tx_{Rcv}$, and we get

   \[ q_{ri}.(D_{Ti} \leftarrow D_i) \leftarrow q_{ri} \cdot \]

   Hence the calculated time for $t(q_{ri})$ becomes

   \[ t(q_{ri}) = t_0 + d_{Q,i} \cdot \]

3. To obtain the sequence of events for transition $rcv_{res}$ for systems $i$ and $j$ the algorithm applies implication rules $Tx_{Rcv}, Tmr_{Exp}, St_{Cr}, Tx_{Rcv}$ in that order, and we get
\[ p_{r,i,j}(D_i \leftarrow D_{T_i}) \leftarrow \text{Res}_j, (D_j \leftarrow D_{T_j}), p_{r,j} \leftarrow q_{r,j}, (D_{T_j} \leftarrow D_j) \leftarrow q_{r}. \]

Hence the calculated time for \( t(p_{r,i,j}) \) becomes

\[ t(p_{r,i,j}) = t_0 + d_{Q,j} + \text{Exp}_j + d_{j,i}. \]

- **Formulating Inequalities:**

Based on the above wanted and unwanted transitions the algorithm avoids transition \texttt{rev..res} while invoking transition \texttt{res..imm} to transit out of \( D_T \). To achieve this, the algorithm automatically derives the following inequality (see Appendix E.1 for more details):

\[ t(p_i) < t(p_{r,i,j}) \quad (8.1) \]

Substituting expressions for \( t(p_i) \) and \( t(p_{r,i,j}) \) previously derived, we get:

\[ d_{Q,i} + \text{Exp}_i < d_{Q,j} + \text{Exp}_j + d_{j,i}. \]

In other words, \( V_t < V_j + d_{j,i} \), where \( V_t = d_{Q,i} + \text{Exp}_i \). \( V_t \) is the time required for system \( i \) to trigger a response transmission (if any).

Alternatively, the system must exist in a state different than \( D_T \) to avoid the unwanted transition, and the algorithm automatically derives the following inequality (see Appendix E.1 for more details):

\[ t(p_{r,i,j}) < t(q_{r,j}). \quad (8.2) \]

Again, substituting expressions derived above, we get:

\[ d_{Q,i} > d_{Q,j} + \text{Exp}_j + d_{j,i}. \]

Note that equations (8.1) and (8.2) are general for any number of responders, where \( i \) and \( j \) are any two responders in the system.

Figure 8.2 shows equations (8.1) and (8.2) in (a) and (b), respectively.
Figure 8.2: Time lines showing possible event sequencing: (a) and (b) sequences do not lead to suppression, while (c) leads to timer suppression.

8.5.1.3 Task specific solutions

- **Topology synthesis**: Given the timer expiration values or ranges, we want to find a feasible solution for the worst-case delays. A feasible solution in this context means positive delay assignments.

In equation (8.1) above, if we take $d_{Q,i} = d_{Q,j}$,

$$
Exp_i - Exp_j < d_{j,i},
$$

The inequalities put an upper limit on the delays $d_{j,i}$, hence, we can always find a positive $d_{j,i}$ to satisfy the inequalities.

Note that, the delays used in the delay matrix reflect delays over the multicast distribution tree. In general, these delays are affected by several factors including the multicast routing protocol, tree type and dynamics, the unicast routing protocol, and the propagation, transmission and congestion delays. One simple topology that

\footnote{The number of inequalities is less than the number of the unknowns ($d_{i,j}$ in this case), hence there are multiple solutions. We can obtain a solution by assigning values to some of the unknowns and solving for the others.}
reflects the delays of the delay matrix is the completely connected network where the underlying multicast distribution tree coincides with the unicast routing.

- **Timer configuration**: Given the delay values or ranges (i.e., bounds), we want to obtain timer expiration values that produce worst-case behavior.

  We can obtain a range for the relative timer settings (i.e., $Exp_i - Exp_j$) using equation (8.1) above.

  Several examples in the section 8.6 illustrate how to apply the above solutions.

  Note, however, that it may not be feasible to satisfy all the constraints, due to upper bounds on the delays for example. In this case the problem becomes one of maximization, where the worst-case scenario is one that triggers maximum number of responses per request. This problem is discussed in Appendix E.2.

8.5.2 Best-case analysis

Best case overhead analysis constructs constraints that lead to maximum suppression, i.e., minimum number of responses.

8.5.2.1 Target event and conditions

The performance criteria is changed to minimize the number of responses per request. The designer identifies the event $q_t$ as the target event, only this time the condition is to avoid (i.e., minimize) the target event. Hence, the algorithm identifies transition $res_{fmr}$ as the unwanted transition. Alternatively, the algorithm identifies transition $recv_{req}$ as the target condition.

8.5.2.2 The search

The following conditions are formulated using steps similar to those given in the worst-case analysis:

$$t(p_{u_i}) > t(p_{r_{i,j}}), \quad (8.3)$$

and

$$t(p_{r_{i,j}}) > t(q_{r_i}). \quad (8.4)$$
These are complementary conditions to those given in the worst case analysis. Figure 8.2 shows equations (8.3) and (8.4) in (c). Refer to the Appendix E.1 for more details on the inequality derivation.

8.5.2.3 Task specific solutions

- **Topology synthesis:**
  
  Given the timer expiration values or ranges, we want to synthesize the best-case delay assignment for the topology.

  Using equation (8.3), if we take \( d_{Q,i} = d_{Q,j} \), we get:

  \[
  Exp_i - Exp_j > d_{i,j}.
  \]

  In this case the output topology may not always be feasible (e.g., if \( Exp_i < Exp_j \)) and problem becomes that of minimization of the number of responses. We discuss this case in Appendix E.2.

- **Timer configuration:**

  Given the topology delay values and ranges, we want to determine the timer expiration settings that produce the best-case behavior.

  From equation (8.4), we get:

  \[
  d_{Q,i} < d_{Q,j} + Exp_j + d_{j,i}.
  \]

  This can be rewritten as,

  \[
  Exp_j > d_{Q,i} - d_{Q,j} - d_{j,i} = -(d_{Q,j} - d_{Q,i} + d_{j,i}).
  \]  

  From equation (8.3) above, we get:

  \[
  d_{Q,i} + Exp_i > d_{Q,j} + Exp_j + d_{j,i},
  \]

  which can be rearranged into
If the delays are given, for example, in an interval $[x,y]$, then, using interval analysis, 
$d_{Q,j} - d_{Q,i} + d_{j,i} = [2x - y, 2y - x] = [a,b]$.
From equations (8.5) and (8.6), we get the two inequalities for the best case performance become:

$$Exp_j > -a, \text{ and}$$
$$Exp_i > Exp_j + b.$$  \hfill (8.6)

We have presented the algorithmic details to construct worst and best case relations, between network delays and timer expiration values, for the protocol overhead in terms of response messages. The solution of the set of inequalities represents the delay and timer settings for the performance stress scenarios. These relations are used in several examples in the next section.

8.6 Example Case Studies

In this section, we present several case studies that show how to apply the previous analysis results to examples in reliable multicast and related protocol design problems.

8.6.1 Topology Synthesis

In this subsection we apply the test synthesis method to the task where the timer values are known and the topology (i.e., D matrix) is to be synthesized according to the worst-case behavior. We explore various timer settings. We use the virtual LAN in Figure 8.3 to look at two examples of topology synthesis, one uses a timers with fixed randomization intervals and the other uses timers that are function of distance.

Let $Q$ be the requester and 1, 2 and 3 be potential responders. At time $t_0$ $Q$ sends the request.

For simplicity we assume, without loss of generality, that the systems are ordered such that $V_{t_i} < V_{t_j}$ for $i < j$ (e.g., system 1 has the least $d_{Q,1} + Exp_1$, then 2, and then 3).

\footnote{From interval analysis: $[x_1, y_1] + [x_2, y_2] = [x_1 + x_2, y_1 + y_2]$, $[x_1, y_1] - [x_2, y_2] = [x_1 - y_2, y_1 - x_2]$, $z > [x, y] \Rightarrow z > y$.}
Figure 8.3: The virtual LAN with 3 potential responders

Thus the inequalities $V_{t_i} < V_{t_j} + d_{j,i}$ are readily satisfied for $i < j$ and we need only satisfy it for $i > j$.

From equation (8.1) for the worst-case above we get:

$$
V_{t_2} < V_{t_1} + d_{1,2},
V_{t_3} < V_{t_1} + d_{1,3},
V_{t_3} < V_{t_2} + d_{2,3}.
$$

(8.7)

By satisfying these inequalities we obtain the delay settings of the worst case topology, as will be shown in the rest of this section.

### 8.6.1.1 Timers with fixed randomization intervals

Some multicast applications and protocols (such as wb, IGMP [Fen97] or PIM [EFH96b]) employ fixed randomization intervals to set the suppression timers. For instance, for the shared white board (wb) [FJL96], the response timer is assigned a random value from the (uniformly distributed) interval $[t, 2^k t]$ where $t = 100$ msec for the source $src$, and 200 msec for other responders.

Assume $Q$ is a receiver with a lost packet. Using wb parameters we get $Exp_{src} = [100, 200]$ msec, and $Exp_i = [200, 400]$ msec for all other nodes.
To derive worst-case topologies from the inequalities (8.7) we may use a standard mathematical tool for linear or non-linear programming, for more details see Appendix E.2. However, in the following we illustrate general techniques that may be used to obtain the solution.

From the inequalities (8.7) we get:
\[ d_{Q,2} + Exp_2 = V_2 < V_1 + d_{1,2} = d_{Q,1} + Exp_1 + d_{1,2}. \]

This can be rewritten as
\[
d_{Q,2} - (d_{Q,1} + d_{1,2}) < Exp_1 - Exp_2 = diff_{1,2},
\]
where
\[
diff_{1,2} = \begin{cases} 
[100,200] - [200,400] = [-300,0] & \text{if 1 is src,} \\
[200,400] - [100,200] = [0,300] & \text{if 2 is src,} \\
[200,400] - [200,400] = [-200,200] & \text{Otherwise.}
\end{cases}
\]

Figure 8.4: The virtual LAN, showing pair-wise delays

Similarly, we derive the following from inequalities for \( V_3 \):
\[ d_{Q,3} - (d_{Q,1} + d_{1,3}) < diff_{1,3}, \text{ and} \\
\[ d_{Q,3} - (d_{Q,2} + d_{2,3}) < diff_{2,3}. \]

As a special case, we assume system 1 to be the source, and for a conservative solution we choose the minimum value of \( diff \), we get:
We then substitute these values in the above inequalities, and assign the values of some of the delays to compute the others.

Example: if we assign $d_{Q,1} = d_{Q,2} = d_{Q,3} = 100\text{msec}$, we get: $d_{1,2} > 300$, $d_{1,3} > 300$ and $d_{2,3} > 200$.

Figure 8.4 shows one possible topology to which the above assigned delays can be applied. These delays exhibit worst-case behavior for the timer suppression mechanism.

### 8.6.1.2 Timers as function of distance

In contrast to fixed timers, this section uses timers that are function of an estimated distance. The expiration timer may be set as a function of the distance to the requester. For example, system $i$ may set its timer to respond to a request from system $Q$ in the interval: $[C_1 \times E_{i,Q}, (C_1 + C_2) \times E_{i,Q}]$, where $E_{i,Q}$ is the estimated distance/delay from $i$ to $Q$, which is calculated using message exchange (e.g. SRM session messages) and is equal to $(d_{i,Q} + d_{Q,i})/2$. (Note that this estimate assumes symmetry which sometimes is not valid.)

[FJL+96] suggests values for $C_1$ and $C_2$ as 1 or $\log_{10} G$, where $G$ is the number of members in the group.

As a special case, we take $C_1 = C_2 = 1$ and attempt to synthesize the worst-case topology. We get the expression

$$Exp_1 - Exp_2 = [(d_{1,Q} + d_{Q,1})/2, d_{1,Q} + d_{Q,1}] - [(d_{2,Q} + d_{Q,2})/2, d_{2,Q} + d_{Q,2}].$$

Example: if we assume that $d_{1,Q} = d_{Q,1} = d_{2,Q} = d_{Q,2} = 100\text{msec}$, we can rewrite the above relation as $Exp_1 - Exp_2 = [-100, 100] \text{msec}$.

Substituting in equation (8.8) above, we get $d_{1,2} > 100\text{msec}$. Under similar assumptions, we can obtain $d_{2,3} > 100\text{msec}$, and $d_{1,3} > 100\text{msec}$.

Topologies with the above delay settings will experience the worst case overhead behavior (as defined above) for the timer suppression mechanism.

As was shown, the inequalities formulated automatically by our method in section 8.5, can be used with various timer strategies (e.g., fixed timers or timers as function of distance). Although the topologies we have presented are limited, a mathematical tool can be used to obtain solutions for larger topologies.
8.6.2 Timer configuration

In this subsection we give simple examples of the timer configuration task solution, where the delay bounds (i.e., D matrix) are given and the timer values are adjusted to achieve the required behavior.

In these examples the delay is given as an interval \([x,y]\) msec. We show examples for worst-case and best-case analysis.

8.6.2.1 Worst-case analysis

If the given ranges for the delays are \([2,200]\) msec for all delays, then the term \(d_{Qj} - d_{Qi} + d_{j,i}\) evaluates to \([-196,398]\). From equation (8.8) above, we get

\[
Exp_i < Exp_j - 196, \text{ to guarantee that a response is triggered.}
\]

If the delays are \([5,50]\) msec, we get:

\[
Exp_i < Exp_j - 45,
\]

i.e., \(i\)'s expiration timer must be less than \(j\)'s by at least 45 msecs.

Note that we have an implied inequality that \(Exp_i > 0\) for all \(i\).

These timer expiration settings would exhibit worst-case behavior for the given delay bounds.

8.6.2.2 Best-case analysis

This case is a direct substitution in equations (8.5) & (8.6) (see Section 8.5.2). For delay ranges of \([2,200]\) msec for all delays, we get \(a = -196\) and \(b = 398\), hence we get \(Exp_j > 196\) msec, and if we take \(Exp_j = 200\), we get \(Exp_i > 598\) msec.

For delay ranges of \([5,50]\) msec for all delays, we get \(a = -40\) and \(b = 95\), hence we get \(Exp_j > 40\) msec, and if we take \(Exp_j = 50\), we get \(Exp_i > 145\) msec.

These timer settings would trigger the best case behavior for the given delay ranges.

Note that for the worst-case analysis we were only able to get relative timer settings, whereas for the best-case analysis we could obtain absolute timer values.

We plan to conduct more intensive studies and simulations to show the utility of our methodology.
8.7 Response Time Analysis

In this section, we conduct the performance analysis with respect to the response time, which is the time for the requester to receive the response and recover from the packet loss. With the assumption of no message loss until recovery, the solution becomes trivial since the response time in that case becomes the time taken for the requester to receive the first response, regardless of other responses. So, for our analysis, we allow the loss of at most a single response message during the recovery phase. Such loss may be selective, i.e., the response may be received by some systems but not others. In this case, transition rules are applied to only those systems that receive the message.

The algorithm obtains possible sequences leading to the target event and calculates the response time for each sequence. To synthesize the worst case scenario that maximizes the response time, for example, the sequence with maximum time is chosen.

8.7.1 Target event

The response time is the time taken by the mechanism to recover from the packet loss, i.e., until the requester receives the response \( p \) and resets its request timer by transitioning out of the \( R_T \) state. In other words, the response time is \( t(p_{rQ}) - t(q_{tQ}) = t(p_{rQ}) - t_0 \).

The designer identifies \( t(p_{rQ}) \) as the target time, hence, \( p_{rQ} \) is the target event.

8.7.2 The search

For illustrative purposes, we present in detail the case of single responder, then discuss the multiple responders case.

- **Backward search**: Starting from \( p_{rQ} \), the backward search yields\(^6\):

\[
p_{rQ}.(R_Q \leftarrow R_{TQ}) \leftarrow p_{tQ}.(D_j \leftarrow D_{Tj}).R_{sj}.R_{TQ} \leftarrow q_{rQ}.(D_{Tj} \leftarrow D_j).R_{TQ}
\]

At which point the algorithm reaches a branching point, where two possible preceding states could cause \( q_{rQ} \). These are the two events in the transition table that cause \( q_{rQ} \):

- The first is transition \( loss \ [q_{tQ}.(R_{TQ} \leftarrow R_Q).D_{Tj}] \) and that ends the backward search for this branch as the initial state \( R_Q \) is reached.

\(^6\)The GFSM may be represented by composition of individual states (e.g., \( State_1, State_2 \) or \( transition_1, State_2 \)).
The second is transition \textit{req.tr} \([\text{Req}_Q.q_t.Q,R_{T_Q},D_j]\). Note that \text{Req}_Q indicates the need for a transition to \(R_{T_Q}\), and the search for this last state yields eventually \(q_t.Q \leftarrow R_{T_Q} . D_j\).

- **Forward search**: The algorithm performs a forward search and checks for consistency of the GFSM.

The forward search step may lead to contradiction with the original backward search, causing rejection of that branch as a feasible sequence. For example, one possible forward sequence from the initial state gives:

\[
q_t.Q \rightarrow R_{T_Q} . D_j = q_r.Q \rightarrow (D_j \rightarrow D_{T_j} . R_{T_Q} = \text{Res}_{T_j} . (D_{T_j} \rightarrow D_j) . p_t . R_{T_Q}
\]

The algorithm then searches two possible next states:

- If \(p_t\) is not lost, and hence causes \(p_r.Q\), then the next state is \(D_j . R_{T_Q}\). But the original backward search started from \(q_t.Q . \text{Req}_Q . R_{T_Q} . D_j\) which cannot be reached from \(D_j . R_{T_Q}\). Hence, we get contradiction and the algorithm rejects this sequence.

- If the response \(p\) is lost by \(Q\), we get \(D_j . R_{T_Q}\) that leads to \(q_t.Q . \text{Req}_Q . R_{T_Q} . D_j\).

The algorithm identifies this as a feasible sequence.

Calculating the response time for each sequence, the algorithm picks the latter sequence as one of max response time.

For **multiple responders**, the algorithm automatically explores the different possible selective loss patterns of the response message. The only feasible sequence obtained by the search is when the requester loses the response and is a sequence in which only one responder (e.g., \(j\)) triggers a response, and the rest suppress. Otherwise, the forward search with single response loss reaches contradiction.

To satisfy this condition, the algorithm creates conditions and inequalities similar to those formulated for the best-case analysis with respect to number of responses (see Section 8.5.2).

### 8.8 Conclusion

We have presented a methodology for test synthesis for performance evaluation of multipoint protocols. In this chapter, our method was applied to evaluate the performance of the
timer suppression mechanism; a common building block for various multipoint protocols. We used a virtual LAN model to represent the underlying network topology.

We adopted the fault-oriented test generation algorithm for search, and extended it to capture timing and delay semantics, and to deal with performance issues for end-to-end multipoint protocols.

Two performance criteria were used for evaluation of the worst and best case scenarios; the number of responses per request, and the response delay. We applied our algorithm to several case studies to illustrate how to use the method in simulation and test synthesis problems relating to real protocols.

We do not claim to have a generalized algorithm that applies to any arbitrary protocol. However, we hope that similar approaches may be used to identify and analyze other protocol building blocks. We believe that such systematic analysis tools will be essential in designing and testing protocols of the future.
Chapter 9

Summary and Future Work

Network protocols are becoming more complex with the growth of the Internet and the introduction of new services, such as multicast. In addition, network failures may cause protocols to behave in an unexpected fashion. In this study, we develop a methodology to study multipoint (multicast-based) protocols, in presence of network failures.

The goal of our methodology is to make network design more robust by systematizing and automating test synthesis for multipoint protocols. We provide a set of practical methods and algorithms to study robustness and worst case performance of Internet multipoint protocols. One major problem we address using our algorithms is the problem of the state space explosion, where the space of possible events and topologies becomes impossible to search exhaustively.

This document presents our methodology to achieve the above goals in the context of multicast routing and end-to-end multipoint protocols. This chapter presents a summary of our contributions, and describes our proposed future work.

9.1 Contributions

In this dissertation we have presented our framework for systematic test synthesis for multipoint protocol design. In this process we have contributed to the development of the methodology, the protocol models and the test generation algorithms. In addition, case studies for multipoint protocols resulted in identification of design errors in the protocols studied.

9.1.1 The Methodology

We have developed the STRESS methodology. Our contributions lie in that we have:
- Proposed a framework for systematic evaluation of multipoint protocol design, through the integration of test generation, simulation, and emulation. The framework can be used to evaluate design trade-offs, analyze protocol behavior under various network conditions, or test protocol implementation.

- Identified test generation as an integral part of the design and testing process of network protocols. Our work is the first work, of which we are aware, that addresses test synthesis—with all its dimensions, topology, event sequences and faults—explicitly and systematically for multipoint network protocols.

9.1.2 Test Generation Algorithms

We have developed three test generation (TG) algorithms: heuristic TG, fault-independent TG, and fault-oriented TG.

- The heuristic approach introduces the notion of representative scenarios to circumvent the state explosion problem. Also, it identifies representative topologies based on equivalence relationships. The equivalence definition suggests that extending the simulated topologies would not reveal additional errors in the protocol. This approach, however, does not automate the generation of the host events and topologies.

- The fault-independent approach uses a forward search technique. The complexity of the algorithm for our case study is reduced from exponential to polynomial in the number of routers by the use of counting equivalence. This approach does not synthesize the topology automatically.

- By contrast, the fault-oriented test generation (FOTG) uses a backward search as the main search technique, starting from the target fault. This approach performs topology synthesis automatically as part of the search process. We have further extended the basic FOTG approach to be used for end-to-end performance evaluation to synthesize worst and best case performance scenarios.

9.1.3 Modeling

We have modeled our target system as an extended global finite state machine (GFSM). The GFSM model used in our methods was extended to capture multicast semantics, timers and delays by introducing the virtual LAN concept. Fault models were integrated into the
system model to include selective packet loss, crashes and extended delays. Performance issues were also addressed in the model to be able to represent criteria such as message overhead and response times.

9.1.4 Case Studies

We have conducted case studies for multicast routing and end-to-end multipoint protocols.

- We have established a set of test-suites for the multicast routing protocols studied, PIM-DM and PIM-SM. Using our method we uncovered several correctness violations in PIM including blackholes caused by the Join, Prune and Graft message loss, Register looping, and wasted bandwidth caused by the Assert mechanism. We also provided detailed simulations for PIM-DM and PIM-SM in ns, and a detailed implementation of PIM-SM.

- We have studied the timer suppression mechanism for end-to-end multipoint protocols. We have synthesized scenarios for best and worst case performance behaviors for the response overhead and response time evaluation criteria. We applied the resulting solution to the topology synthesis and timer configuration tasks.

9.2 Future Work

Our future work includes improvements of current methods and algorithms, in addition to exploring potential extensions and applications of the methodology.

- Improving current algorithms

  - The fault-independent test generation method, as presented in this study, is not able to synthesize the topology as part of the output test scenarios. Symbolic representation techniques may be investigated to add the topology synthesis capability to the method.

  - The automatic identification of equivalence classes remains part of future work.

  - Completeness of topology synthesis for fault-oriented test generation should be proven in order to achieve full coverage of the state space. Using symbolic representation, the topologies synthesized by FOTG may be proven to be sufficient to capture all the robustness violations in the given protocol.
- In order to reduce the complexity of the search algorithms in general, and FOTG in specific, reduction techniques based on equivalence, early detection of reachability, or others should be investigated.

- **Multi-hop topologies**
  The methods in this study use a single-hop LAN or a virtual LAN model. However, to enable studies involving more complex topologies, or clusters of systems, the model must be extended to represent multi-hop LANs. Clusters of networks may be represented as multi-hop virtual LANs. In order to enable multi-hop studies we must consider inter-LAN interactions, in addition to interactions between systems on the same LAN.

- **Trace driven test generation**
  Traces collected from operational networks could be used to direct the process of test generation. Fault models that occur frequently in real networks can be targeted by the search algorithm, for example. Also, traffic, membership, topology and loss models can be used to direct the search. For example, during the search process, higher priority may be given to scenarios that occur more frequently.

- **Mapping functional correctness into the model**
  In general, the criteria of protocol correctness are given at the functional level, such as single packet delivery, or absence of duplicates. These criteria must be mapped into the model used by the search algorithm in order to check for errors. This mapping must be proven to be sufficient (for completeness) and necessary (to avoid false alarms), to cover all the functional errors and only indicate erroneous scenarios. Future work should consider automating the process of this mapping, or at least systematize the proof process.

- **Implementation of algorithms and integration with simulation**
  Simulation is a valuable tool for designing and evaluating network protocols. Researchers usually use their insight and expertise to develop simulation inputs and test suites. Our method may be used to assist in automating the process of choosing simulation inputs and scenarios.

  Our future work will include implementing a more complete tool to automate our method (including search algorithms and modeling semantics) and tie it to a network simulator to be applied to a wider range of multipoint protocols.
• Validating protocol building blocks

The design of new protocols and applications often borrows from existing protocols or mechanisms. Hence, there is a good chance of re-using established mechanisms, as appropriate, in the design process. Identifying, verifying and understanding building blocks for such mechanisms is necessary to increase their re-usability. Our method may be used as a tool to improve that understanding in a systematic and automatic manner.

Ultimately, one may envision that a library of these building blocks will be available, from which protocols (or parts thereof) will be readily composable and verifiable using CAD tools; similar to the way circuit and chip design is carried out today using VLSI design tools.

In our work we have identified some building block mechanisms for multicast routing and end-to-end multipoint protocols, namely, the Join/Prune mechanism and the timer suppression mechanism. More work is needed to identify more building blocks to cover a wider range of protocols and mechanisms.

A related research area is to classify networking problems (e.g., timing problems) into categories and identify mechanistic building blocks that deal with different kinds of problems.

• Application to protocol comparison and benchmarking

Different protocols may be grouped in the same class based on their functionality. The absence of a common test suite for a given class of protocols makes it hard to compare these protocols in a neutral way.

Our method may be extended to develop tests that would functionally stress protocols with respect to given criteria, thus enabling the establishment of common test suites. These tests would be used to compare different protocols within the same class. Also, they may be used to test different versions, refinements or implementations of the same protocol, and hence facilitate interoperability testing.

• Design space exploration and sensitivity analysis

Protocol design usually entails the configuration and setting of various parameters. Changing these parameters may lead to change in behavior or performance. It is often prohibitively complex to investigate such parameter design space manually, or exhaustively.
This problem may be alleviated by the use of automatic generation of test suites for given parameter sets. For example, examining how the worst-case behavior or scenarios change with parameter values, may help in making better design trade-offs.

This kind of sensitivity analysis may also be carried out for environment parameters, such as delay, bandwidth or membership distribution. Changing these parameters may change the behavior of the protocol in some respect. Our method may be used to tune protocol parameters automatically for best case behaviors, as we have shown for the case of the timer configuration task.

In our study, we have investigated only a single point in the design space at a time. Sensitivity analysis investigates a broader spectrum of parameter values. Thus, it may require more efficient algorithms and ways of filtering and processing of the output results in a manner useful to the researcher.

- **Generalization to performance bound analysis**

  An approach similar to the one we have taken for end-to-end performance evaluation in Chapter 8 may be based on some performance bounds, instead of worst or best case analyses. We call such approach ‘condition-oriented test generation’ or analysis.

  For example, a target event may be defined as ‘the response time exceeding certain delay bounds’ (either absolute bounds or as a function of some parameter). If such a scenario is not feasible, that indicates that the protocol gives absolute guarantees (under the assumptions of the study). This may be used to design or analyze quality-of-service or real-time protocols, for example.

- **Applicability to other problem domains**

  So far, our method has been applied to case studies on multicast routing robustness and multipoint protocol performance evaluation, in the context of the Internet.

  Other problem and application domains may introduce new mechanistic semantics or assumptions about the system or environment. One example of such domains includes sensor networks. These networks, similar to ad-hoc networks, assume dynamic topologies, lossy channels, and deal with stringent power constraints, which differentiates their protocols from Internet protocols [EGH99].

  Possible research directions in this respect include

  - extending the topology representation or model to capture dynamics, where delays vary with time,
– defining new evaluation criteria that apply to the specific problem domain, such as power usage, and
– investigating the algorithms and search techniques that best fit the new model or evaluation criteria.
Appendix A

Heuristic Test Generation for PIM-SM

In this appendix we present our case study for PIM-SM using the heuristic test generation approach (see Chapter 4). This study uses the same equivalent topologies, representative scenarios and test suites that were used for the case study on PIM-DM presented earlier for the heuristic approach.

An overview of PIM-SM is given first. Then we present an elaborate example of applying the heuristic approach in conjunction with simulation, followed by the detailed results of the case study.

A.1 PIM-SM Overview

PIM-SM is a multicast routing protocol that uses explicit join mechanisms for building shared multicast trees. For simplicity, we do not address source-specific trees in this description.

![Diagram of PIM-SM rendezvous](image)

Figure A.1: How senders rendezvous with receivers
As shown in figure A.1, when a receiver’s local router (A) discovers it has local receivers, it starts sending periodic join messages toward a group-specific Rendezvous-Point (RP). The join messages are multicast hop-by-hop. Each router along the path toward the RP builds a wildcard (any-source) route entry for the group and sends the join messages on toward the RP. A route entry is the state held in a router to maintain the distribution tree. Typically it includes the source address, group address, the interface from which packets are accepted (incoming interface), and the list of interfaces to which packets are sent (outgoing list). This state forms a shared, RP-rooted, distribution tree that reaches all group members.

When a source first sends to a group, its local router (D) unicasts register messages to the RP with the source’s data packets encapsulated within. Data packets reaching the RP are forwarded natively down the shared tree toward group members.

Similarly, when a member leaves the group, a prune message is sent by the local router, to stop the multicast traffic from flowing down the branch leading to the pruned member.

Being robust to (at least) a single message loss, even in the presence of unicast inconsistencies, was a design goal for PIM-SM. The Assert and prune-override mechanisms for PIM-SM are the same as those presented earlier for PIM-DM.

A.2 Test suites

The topologies used for the study are those shown in figure A.2.

The simulation environment and tracing semantics are the same as those given in Section 4.2.2.

Subsetting For brevity, we do not consider source-specific trees and switching to the shortest paths in this paper. This is an example of state subsetting, since we consider shared group states while disregarding source-specific states.

The messages considered in the study are join, prune, assert and register messages. To study joins, prunes and asserts without the effect of registers, we consider a topology where the source and the RP are co-located (see S1 in figure 4.4, topology 1). This is an example of message subsetting.

When studying registers, joins and prunes we consider topology 2 in figure 4.4 where: (a) S2 is the source, hence node A sends registers to the RP, and (b) the routed topology has consistent unicast routing, to eliminate the effect of the assert mechanism. This represents function (or mechanism) subsetting. Only triggered actions are investigated for simplicity.
A.3 Applying the Method

We have implemented and a detailed implementation of PIM-SM\(^1\). The method is applied in a manner similar to that presented in Section 4.3.

**Example** In our simple example, an error condition is any packet loss or duplication, experienced by the end-points. A faulty scenario (without packet loss) that leads to two error conditions, is identified and explained. Then the protocol actions leading to the errors are analyzed.

The representative scenario explained here is ‘J1:J2:L1:L2’ using topology 1. This scenario was identified automatically as a faulty scenario. Traces in figure A.3 give the history of the errors found. A trace takes the following format: ‘**R2 Node A Rcv 7 t 190**’ meaning that receiver **R2** in node **A** received a data packet with sequence number **7** at time **190ms** from the beginning of the simulation run. The first error (i.e. the packet duplication) has the host event ‘J2’ as the closest join or leave host event in its history at time 200ms. The error is a join transient caused by parallel paths to the RP. The error

---

\(^1\)Our detailed PIM-SM simulation mimics the unix ‘pimd’ [Hei97] implementation model, and hence is able to capture many implementation aspects. We plan to develop an interface between the simulator and an operational network running the pimd code. However, the analyses presented in this study are based strictly on the protocol specification, independent of the implementation.
Figure A.3: Simple packet trace graph showing packet loss and duplication

is resolved using the Assert messages exchanged during the duplication at time 246ms. The second error (i.e. packet loss) is a leave transient; it has a host event ‘L1’ in its recent history. The loss is due to the prune sent by node A at 300ms, and is resolved by a prune-override sent by node B at 310ms.

Although the protocol actions leading to the end-point errors (specified as any packet loss or duplication in this specific example) are considered transient errors, they are not considered protocol design errors. We do, however, address protocol design errors in Section A.5.

A.4 Scenario and protocol coverage

While the fact that we were able to discover design errors provides some evidence of the method’s utility, we would like to quantify the coverage of protocol states and possible scenarios.

The overall protocol coverage has two dimensions. The first is the protocol state coverage, and we attempt to cover this dimension using the representative scenarios’ reachable states. Investigation of the loss scenarios does not affect protocol coverage significantly.

The second dimension is the space of possible interaction scenarios between the state machines (in different routers) within the topology. This dimension is explored by investigating the selective loss scenarios.
**Scenarios covered**  The initial number of simulated scenarios *without* protocol message loss was:

\[ \sum_{\text{topologies}} (\text{No. rep. scenarios}) \]

Where No. rep. scenarios is the number of *representative* scenarios, equal to 6 in our case (discussed in section 4.1.1.2), and the topologies are the two discussed in section 4.2.2. Hence we simulated 12 scenarios without protocol message loss.

After feeding back the link traces for the messages under study, the loss patterns were assigned to the corresponding links. The scenario generator then set-up the simulations for the new scenarios with loss.

The total number of scenarios *with* protocol message loss simulated is given by the following formula:

\[ \sum_{\forall \text{Topos}} \left( \sum_{\forall \text{Reps}} \left( \sum_{\forall \text{Msgs}} \left( \sum_{\forall \text{Links}} \text{LinkMsgs} \cdot 2^{\left( \text{LinkRtrs} - 1 \right)} \right) \right) \right) \]

where the terms used are described in the following table.

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topos</td>
<td>Topologies</td>
</tr>
<tr>
<td>Reps</td>
<td>Representative Scenarios</td>
</tr>
<tr>
<td>Msgs</td>
<td>Messages under study</td>
</tr>
<tr>
<td>LinkMsgs</td>
<td>No. messages traversing the link</td>
</tr>
<tr>
<td>LinkRtrs</td>
<td>No. routers connected to the link</td>
</tr>
</tbody>
</table>

For each topology, this formula gives the number of scenarios automatically generated after the first simulation run, during which the number of messages and links (traversed by these messages) is counted.

For example, for the first topology, the messages under study were *joins*, *prunes* and *asserts*. The representative scenarios triggered 16 *joins*, 12 *prunes*, and 12 *asserts* on the LAN, and 16 *joins* and 16 *prunes* on point-to-point links. For the second topology, the messages under study were *joins* and *prunes*. The representative scenarios triggered 16 *joins*, and 18 *prunes* on the LAN, and 6 *joins* and 18 *prunes* on point-to-point links. Hence, the total number of scenarios with loss became 352 and 296 scenarios, respectively.
Protocol code coverage  A large portion of the multicast support code in NS was annotated automatically to provide code tracing. The representative scenarios without loss invoked 84 procedures out of 91 overall annotated procedures. The procedures that were not invoked dealt mainly with source-specific state (which was abstracted in our test suites), or with the modularity of the object-oriented nature of the code.

A.5 Results

This section describes the protocol design errors uncovered for PIM-SM under STRESS.

We modified the error conditions to avoid join and leave transients, since, unlike our simple example above, we are only interested in design errors. The new error conditions do not consider single duplication or loss.

Following is a summary of the major faulty scenarios encountered, and how they relate to STRESS. For a more detailed discussion of the protocol errors and fixes see section A.5.2.

A.5.1 Summary of Results

We describe a partial list of faulty scenarios captured by STRESS. We obtained this list after simulating only a few of the representative scenarios. The traces produced provided guidance to discover the protocol errors. Design errors discovered include Assert, Join/Prune and Register mechanisms.

Asserts  For the first topology (figure 4.4, topology 1), a black hole was observed for one receiver.

The faulty scenario in this case involved another receiver joining in the recent history of the black hole. By analyzing the protocol trace history after rolling back, we noticed that an Assert process took place right before the loss.

In addition, the faulty scenario included the loss of a join message, which prevented the establishment of the branch of the shared tree from the Assert winner to the RP. Hence, the protocol design error is allowing a router on a branch of the tree that is not completely established, to participate in Asserts.

Joins and Prunes  Over the same topology (i.e. figure 4.4, topology 1), several other faulty scenarios lead to black holes. The host scenarios involved one receiver leaving just
before black holes were experienced by the other receiver. In these cases join and prune messages occurred the recent history of the end-point error.

Furthermore, all such scenarios included either: (i) loss of a join message, preventing a pruned branch from being re-established; or (ii) selective loss of a prune message, preventing a join (i.e. prune-override) from being triggered. The protocol design error in this case was not allowing a second chance for routers with downstream members to override prunes.

**Registers** In the second topology (figure 4.4, topology 2), faulty scenarios were captured that cause packet duplicates at the end-points.

In this case, the observed faulty scenarios did not follow a regular pattern, and were developed iteratively (i.e. when one faulty scenario led to a suggested fix in the protocol, the fix was implemented and the method re-run to observe further faulty scenarios).

The first scenario involved a single host receiving duplicates merely by joining the group. The packets were being delivered at least twice, once directly from the source –by virtue of being on the same LAN–, and the second delivery from the shared tree after the register reached the RP and was sent down the shared tree. When the number of packet duplicates exceeded two, this suggested a loop. The loop occurred when a packet received over the shared tree on the LAN, was (a) picked up by the local router, (b) re-registered to the RP, and (c) forwarded down the shared tree again. The protocol error was allowing the packets to flow down from the shared tree to the originating LAN, and be re-registered. The fix was to prune such sources from the shared tree.

The second scenario involved another receiver joining before the duplicates were observed. The pruned branch of the shared tree was re-established by the joining receiver, allowing the packets to flow down the shared tree to the originating LAN, and subsequently, causing the loop.

The third scenario involved a prune message loss, again allowing the packets to flow down the shared tree to the originating LAN, and led to looping.

Rules were added to prevent packets from being forwarded back on their original LANs in the above scenarios.
A.5.2 Detailed Results

The rest of this section describes the above faulty scenarios in more detail, and illustrates how the solutions were developed with the aid of STRESS. After the solutions were integrated into the protocol simulator, we applied ‘regression testing’ to verify that the fixes did not introduce any new errors.

A. Assert analysis

Following is a discussion of the pathological cases found in the Assert process. An exhaustive list of the results is not included in this document for brevity. A few errors in the PIM-SM specification were unveiled during this process, we focus on errors that created the possibility of packet loss (i.e. black holes).

The scenario

In this scenario, the topology in figure A.4 was set-up such that A’s next-hop towards the RP is C, and B’s next-hop towards the RP is D.

Consider the sequence of events shown in figure A.4, which used the representative scenario ‘J1:J2:L1:L2’ with the loss of a join message on the link between C and RP.

During the last two events of the scenario (steps 6 and 7), D loses the Assert process to C (with lower metric or higher address). Subsequently, D removes the LAN from its entry’s interface list, and R1 stops receiving packets from S1. This problem persists until/unless the branch of the tree from C to RP is established.

Figure A.4: The Assert scenario under study
Discussion and fix The current rules of the PIM specification aim to guarantee ‘at-most’ one forwarder on a multi-access network. However, to ensure proper delivery of packets without packet loss, the right semantics should be ‘exactly’ one forwarder.

The problem arises, more specifically, because the PIM specification does not distinguish between an ‘active’ entry (i.e. an entry created due to arrival of data packets, e.g. a multicast forwarding cache), and an entry on a branch of a tree that is not yet established (or an ‘inactive’ entry). An ‘inactive’ entry may win an Assert process, resulting in black holes.

To solve this problem, we modified the specification to ensure ‘exactly’ one forwarder semantics using the following rule: A router receiving a data packet (or Assert) on an outgoing interface of a matching entry does not participate in the Assert process unless the entry is ‘active’. Figure A.5 illustrates the ‘ActiveState’ added to the transition diagram to realize the solution.

B. Join/Prune analysis
In this analysis we address the effect of selective loss of Join/Prune messages. Although this problem has been addressed in recent releases of the PIM-SM specification, we provide a more efficient solution.

We use the topology given in figure A.6, I. The representative scenario used is ‘J1:J2:L1:L2’ with the second join from node A lost on the LAN.

We assume that S1 sends packets to group G throughout the simulation. Consider the sequence of events given in figure A.6, I. After the last event (step 5), R2 stops receiving

Figure A.5: Transition diagram for joins and asserts
1) R1 joins the group. B sends joins towards RP.
2) R2 joins the group. A sends joins towards RP.

3) R1 leaves the group. B multicasts prunes onto the LAN.
4) A gets the prune and sends a join to override. The join is lost.
5) C gets the prune and sends it towards RP.

3) R1 leaves the group. B multicasts prunes onto the LAN.
4) A does not receive the prune, and so does not override.
5) C gets the prune and sends it towards RP.

Figure A.6: The Join/Prune scenario under study

S1’s packets. This problem persists until A sends the next periodic join to C and re-establishes the pruned branch of the tree. A similar problem is encountered in figure A.6, II, when the prune sent from B is selectively lost on the LAN by A and received by C.

Discussion and fix

The solution suggested by the PIM specification introduces a deletion timer. This, however, increases the leave latency, and incurs unnecessary data overhead.

A more efficient solution would be to have the upstream router (C) announce a ‘prune-alert’, before removing the LAN from its outgoing list, by resending the prune message previously received from B.

C. Register analysis

Following is a description of the scenarios that exhibit packet duplication due to register messages, and the suggested fixes to eliminate such duplication. The fixes were applied iteratively, until the error was eliminated.

i. First scenario (single source, single receiver):

In this scenario we consider S2 and R2 in figure A.7 (I). Consider the sequence of events in the figure.

Packet duplication and register looping occur in the above scenario. A similar scenario occurs when R2 joins first then S2 starts sending to the group.

Suggested fixes
The required behavior is to send a -triggered and periodic- source-specific prune off of the shared tree, if a router has source-specific state for registering and shared tree state for the same group (regardless of the incoming interface settings).

ii. Second scenario (single sender, two receivers):

We assume the implementation of the above fixes to the simulator, then consider the sequence of events in figure A.7 (II). This scenario exhibits packet duplication and register looping.

Suggested fix

The problem arises because the packets are forwarded back on the originating LAN, and treated as if they were new packets originated by the directly connected source. The following rule solves this problem for the given scenario:

- A router receiving join message must NOT add an interface on the same subnet as a source S, for any source specific entry for S associated with same group.

iii. Third scenario (single source, single receiver with message loss):

Considering the scenario in figure A.7 (III).

The source specific prune sent from A to C –when A having a shared tree state, creates the source specific entry for registering– is lost.

Packet duplication and register looping problems are experienced in this scenario. The problem persists until a periodic Join/Prune message is successfully sent upstream.

Suggested fix
To be robust to (at least) one message loss, we suggest the following rule for packet forwarding:

- A router must **NOT** forward a packet onto the subnet from which the packet was originated. This is achieved by performing a check on the source and the outgoing interface before building a source specific state or before forwarding a packet\(^2\).

\(^2\)Most implementations create a cache for forwarding packets. This check can be done only once when creating the cache, and is not done per packet.

This is different than the 'incoming interface' check stated by the current specification. In the specific case discussed here, the looping multicast packets will match on the incoming interface (the LAN) for the source-specific entry.
Appendix B

State Space Complexity

In this appendix we present analysis for the state space complexity of our target system. In specific we present completeness proof of the state space and the formulae to compute the size of the correct state space.

B.1 State Space Completeness

We define the space of all states as $X^*$, denoting zero or more routers in any state. We also define the algebraic operators for the space, where

$$X^* = X^0 \cup X^1 \cup X^{2+}$$  \hspace{1cm} (B.1)

$$(Y^n, X^*) = (Y^{n+}, \{X - Y\}^*)$$  \hspace{1cm} (B.2)

B.1.1 Error states

In general, an error may manifest itself as packet duplicates, packet loss, or wasted bandwidth. This is mapped onto the state of the global FSM as follows:

1. The existence of two or more forwarders on the LAN with one or more routers expecting packet from the LAN (e.g., in the $NH_X$ state) indicates duplicate delivery of packets.

2. The existence of one or more routers expecting packets from the LAN with no forwarders on the LAN indicates a deficiency in packet delivery (join latency or black holes).
3. The existence of one or more forwarders for the LAN with no routers expecting packets from the LAN indicates wasted bandwidth (leave latency or extra overhead).

- for duplicates: one or more $NH_X$ with two or more $F_X$;

\[(NH_X, F_X^{2+}, X^*)\] (B.3)

- for extra bandwidth: one or more $F_X$ with zero $NH_X$;

\[(F_X, \{X - NH_X\}^*)\] (B.4)

- for blackholes or packet loss: one or more $NH_X$ with zero $F_X$;

\[(NH_X, \{X - F_X\}^*)\] (B.5)

**B.1.2 Correct states**

As described earlier, the correct states can be described by the following rule:

\[\exists\text{ exactly one forwarder for the LAN } \iff \exists\text{ one or more routers expecting packets from the LAN.}\]

- zero $NH_X$ with zero $F_X$;

\[\{(X - NH_X - F_X)^*\}\] (B.6)

- one or more $NH_X$ with exactly one $F_X$;

\[(NH_X, F_X, \{X - F_X\}^*)\] (B.7)

from (B.2) and (B.3) we get:

\[(NH_X, F_X^{2+}, \{X - F_X\}^*)\] (B.8)

if we take the union of (B.8), (B.5) and (B.7), and apply (B.1) we get:
\[(NH_X, X^*) = (NH_X^{1+}, \{X - NH_X\}^*)\]  \hspace{1cm} (B.9)

also, from (B.4) and (B.2) we get:

\[(F_X^{1+}, \{X - NH_X - F_X\}^*)\]  \hspace{1cm} (B.10)

if we take the union of (B.10) and (B.6) we get:

\[(F_X^s, \{X - NH_X - F_X\}^*) = (\{X - NH_X\}^*)\]  \hspace{1cm} (B.11)

taking the union of (B.9) and (B.11) we get:

\[(NH_X^s, \{X - NH_X\}^*) = (X^*)\]  \hspace{1cm} (B.12)

which is the complete state space.

\section*{B.2 Number of Correct and Error State Spaces}

\subsection*{B.2.1 First case definition}

For the correct states: \((\{X - NH - F\}^*)\) reduces the symbols from which to choose the state by 2; i.e. yields the formula:

\[C(n + (s - 2) - 1, n) = C(n + s - 3, n).\]

While \((NH, F, \{X - F\}^*)\) reduces the number of routers to choose by 2 and the number of symbols by 1, yielding:

\[C((n - 2) + (s - 1) - 1, n - 2) = C(n + s - 4, n - 2).\]

\subsection*{B.2.2 Second case definition}

For the correct states: \((\{X - NH_X - F_X\}^*)\) reduces, the number of states by 4, yielding
While \((NH_X, F_X, \{X - F_X\}^*)\) reduces the number of routers to \(n-2\) and the symbols to \(s-2\) and yields

\[
4 \cdot C((n - 2) + (s - 2) - 1, n - 2) = 4 \cdot C(n + s - 5, n - 2).
\]

We have to be careful here about overlap of sets of correct states. For example \((NH, F, \{X - F_X\}^*)\) is equivalent to \((NH_{Rtx}, F, \{X - F_X\}^*)\) when a third router is in \(NH_{Rtx}\) in the first set and \(NH\) in the second set. Thus we need to remove one of the sets \((NH, F, NH_{Rtx}, \{X - F_X\}^*)\), which translates in terms of number of states to

\[
C((n - 3) + (s - 2) - 1, n - 3) = C(n + s - 6, n - 3).
\]

A similar argument is given when we replace \(F\) above by \(F_{Det}\), thus we multiply the number of states to be removed by 2. Thus, we get the total number of equivalent correct states:

\[
C(n + s - 5, n) + 4 \cdot C(n + s - 5, n - 2) - 2 \cdot C(n + s - 6, n - 3).
\]

To obtain the \textit{ErrorStates} we can use:

\[
\text{ErrorStates} = \text{TotalStates} - \text{CorrectStates}.
\]
Appendix C

Forward Search Algorithms

This appendix includes detailed procedures that implement the forward search method as described in Chapter 6. It also includes detailed statistics collected for the case study on PIM-DM.

C.1 Exhaustive Search

The **ExpandSpace** procedure given below implements an exhaustive search, where \( W \) is the working set of states to be expanded, \( V \) is the set of visited states (i.e. already expanded), and \( E \) is the state currently being explored. Initially, all the state sets are empty. The `nextState` function gets and removes the next state from \( W \), according to the search strategy; if depth-first then \( W \) is treated as a stack, or as a queue if breadth-first.

Each state is expanded by applying the stimuli via the ‘forward’ procedure that implements the transition rules and returns the new stable state \( New \).

```plaintext
ExpandSpace(initGState){
  add initGState to W
  while W not empty {
    E = nextGState from W;
    add E to V;
    \forall state \in E
      \forall stim applying to state {
        New = forward(E,stim);
        if New \notin W or V
          add New to W;
      }
  }
}
```

116
The initial state $initGState$ may be generated using the following procedure, that produces all possible combinations of initial states $I.S.$.

$Init(depth,GState)\{$
\begin{itemize}
  \item $\forall state \in I.S. \{$
    \begin{itemize}
      \item add state to $GState$;
      \item $depth = depth - 1$;
      \item if $depth = 0$
        \begin{itemize}
          \item $ExpandSpace(GState)$;
        \end{itemize}
      \item else
        \begin{itemize}
          \item $Init(depth, GState)$;
          \item remove last element of $GState$;
        \end{itemize}
    \end{itemize}
\end{itemize}
$}\}$

This procedure is called with the following parameters: (a) number of routers $n$ as the initial $depth$ and (b) the $emptystate$ as the initial $GState$. It is a recursive procedure that does a tree search, depth first, with the number of levels equal to the number of routers and the branching factor equal to the number of initial state symbols $|I.S.| = i.s.$. The complexity of this procedure is given by $(i.s.)^n$.

**C.2 Reduction Using Equivalence**

We use the counting equivalence notion to reduce the complexity of the search in 3 ways:

1. The first reduction we use is to investigate only the equivalent initial states, we call this algorithm $Equiv.$.

   One procedure that produces such equivalent initial state space is the $EquivInit$ procedure given below.

   $EquivInit(S,i,GState)\{$
   \begin{itemize}
     \item $\forall state \in S$
     \begin{itemize}
       \item for $j = i$ to 0 {
         \begin{itemize}
           \item $New = emptystate$;
         \end{itemize}
       \end{itemize}
     \end{itemize}
   \end{itemize}
$\}$
<table>
<thead>
<tr>
<th>Rtrs</th>
<th>Exhaustive</th>
<th>Equiv</th>
<th>Equiv+</th>
<th>Reduced</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>1.555556</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>24</td>
<td>18</td>
<td>18</td>
<td>2.888889</td>
</tr>
<tr>
<td>3</td>
<td>178</td>
<td>52</td>
<td>30</td>
<td>30</td>
<td>5.933333</td>
</tr>
<tr>
<td>4</td>
<td>644</td>
<td>114</td>
<td>48</td>
<td>48</td>
<td>13.41666</td>
</tr>
<tr>
<td>5</td>
<td>2176</td>
<td>238</td>
<td>73</td>
<td>73</td>
<td>29.80822</td>
</tr>
<tr>
<td>6</td>
<td>7480</td>
<td>496</td>
<td>106</td>
<td>106</td>
<td>70.56604</td>
</tr>
<tr>
<td>7</td>
<td>24362</td>
<td>1004</td>
<td>148</td>
<td>148</td>
<td>164.6081</td>
</tr>
<tr>
<td>8</td>
<td>80830</td>
<td>2037</td>
<td>200</td>
<td>200</td>
<td>404.15</td>
</tr>
<tr>
<td>9</td>
<td>259270</td>
<td>4081</td>
<td>263</td>
<td>263</td>
<td>985.8175</td>
</tr>
<tr>
<td>10</td>
<td>843440</td>
<td>8198</td>
<td>338</td>
<td>338</td>
<td>2495.385</td>
</tr>
<tr>
<td>11</td>
<td>2684665</td>
<td>16386</td>
<td>426</td>
<td>426</td>
<td>6302.031</td>
</tr>
<tr>
<td>12</td>
<td>8621630</td>
<td>32810</td>
<td>528</td>
<td>528</td>
<td>16328.84</td>
</tr>
<tr>
<td>13</td>
<td>27300731</td>
<td>65574</td>
<td>645</td>
<td>645</td>
<td>42326.71</td>
</tr>
<tr>
<td>14</td>
<td>86885238</td>
<td>131180</td>
<td>778</td>
<td>778</td>
<td>111677.7</td>
</tr>
</tbody>
</table>

Figure C.1: Simulation statistics for forward algorithms. ExpandedStates is the number of visited states.

for \(k = 0\) to \(j\)

add state to New;

\[\text{New} = \text{New} \cdot \text{GState}\]

\[\tilde{S} = \text{trunc}(S, \text{state});\]

if \((i - j) = 0\)

ExpandSpace(\text{New});

else

EquivInit(\(\tilde{S}, i - j, \text{New}\));

\}

\}

This procedure is invoked with the following parameters: (a) the initial set of states \(I, S\) as \(S\), (b) the number of routers \(n\) as \(i\), and (c) the empty state as \(\text{GState}\). The procedure is recursive and produces the set of equivalent initial states and invokes the ExpandSpace procedure for each equivalent initial state. The ‘trunc’ function truncates \(S\) such that \(\tilde{S}\) contains only the state elements in \(S\) after the element \(\text{state}\). For example, \(\text{trunc}([F, NM, M], F) = \{NM, M\}\).
2. The second reduction we use is during state comparison. Instead of comparing the actual states, we compare and store equivalent states. Hence, the line ‘if New $\not\in W$ or $V$’ would check for equivalent states. We call the algorithm after this second reduction `Equiv+`.

3. The third reduction is made to eliminate redundant transitions. To achieve this reduction we add flag check before invoking `forward`, such as stateFlag. The flag is set to 1 when the stimuli for that specific state have been applied. We call the algorithm after the third reduction the reduced algorithm.

### C.3 Complexity analysis of forward search for PIM-DM

The number of reachable states visited, the number of transitions and the number of erroneous states found were recorded. The result is given in figures C.1, C.2, C.3, C.4. The reduction is the ratio of the numbers obtained using the exhaustive algorithm to those obtained using the reduced algorithm.

The number of expanded states denotes the number of visited stable states and is measured simply as the number of states in the set $V$ in ‘ExpandSpace’ procedure. The number of forwards is the number of times the ‘forward’ procedure was called denoting the number of transitions between stable states. The number of transitions is the number of
Figure C.3: Simulation statistics for forward algorithms. Transitions is the number of transient states visited. Visited transient states that are increased with every new state visited in the ‘forward’ procedure. The number of error states is the number of stable (or expanded) states violating the correctness conditions.

The number of transitions is reduced from $O(4^n)$ for the exhaustive algorithm to $O(n^4)$ for the reduced algorithm. This means that we have obtained exponential reduction in complexity, as shown in figure C.5.
Figure C.4: Simulation statistics for forward algorithms. The number of stable error states reached.

Figure C.5: Reduction ratio from exhaustive to the reduced algorithm
Appendix D

FOTG Algorithms

This appendix includes pseudo-code for procedures implementing the fault-oriented test generation (FOTG) method presented in Chapter 7. In addition, it includes detailed results of our case study to apply FOTG to PIM-DM.

D.1 Pre-Conditions

The ‘conds’ array contains the post-conditions (i.e., the effects of the stimuli on the system) and is indexed by the stimulus. The ‘stimulus’ function returns the stimulus (if any) of the condition. The ‘transition’ function returns the transition or state of the condition 1. The pre-conditions are stored in an array ‘preConds’ indexed by the stimulus.

\[
\begin{align*}
\text{PreConditions} & \{ \\
& \forall \text{stim} \in \tau \\
& \hspace{0.5cm} \forall \text{cond} \in \text{conds}[\text{stim}] \{ \\
& \hspace{1cm} s = \text{stimulus}(\text{cond}); \\
& \hspace{1cm} t = \text{transition}(\text{cond}); \\
& \hspace{1cm} \text{add } t.\text{stim to } \text{preConds}[s]; \\
& \} \\
& \}
\end{align*}
\]

D.2 Dependency Table

The ‘dependencyTable’ procedure generates the dependency table depTable from the transition table of conditions conds.

---

1If there’s a state in the condition, this may be viewed as state ⇒ state transition, i.e., transition to the same state.
dependencyTable{
\forall stim \in \tau
\forall cond \in conds[stim] 
  \begin{align*}
  endState &= end(cond); \\
  startState &= start(cond); \\
  \text{add} \ startState, stim \text{ to depTable[endState]};
  \end{align*}
}

For each state s, that is endState of a transition, a set of startState - stimulus pairs leading to the creation of s is stored in the depTable array. For s \in I.S. a symbol denoting initial state is added to the array entry. For our case study I.S. = \{NM, EU\}.

D.3 Topology Synthesis

The following procedure synthesizes minimum topologies necessary to trigger the various stimuli of the protocol. It performs the third and forth steps explained in Section 7.2.

buildMinTopos(stim){
\forall cond \in preConds[stim]
\begin{align*}
  st &= end(cond); \\
  stm &= stimulus(cond); \\
  \text{if type}(stm) = orig \\
      \text{add} \ st \text{ to MinTopos[stim]};
  \text{else} \\
  \text{if } \exists \text{Topo}(stm) \\
  \text{buildMinTopos(stm)};
  \forall topo \in MinTopos[stim] \\
  \text{add} \ st \text{ to MinTopos[stim]}; \\
\end{align*}
}

D.4 Backward Search

The ‘Backward’ procedure calls the ‘Rewind’ procedure to perform the backward search. A set of visited states V is kept to avoid looping. For each state in GState possible backward
implications are attempted to obtain valid backward steps toward initial state. ‘Backward’ is called recursively for preceding states as a depth-first search. If all backward branches are exhausted and no initial state was reached the state is declared unreachable.

\[\text{Backward}(GState)\{
\]
  \text{if } GState \in V \text{ return loop}
  \text{add } GState \text{ to } V
  \forall s \in GState\{
    bkwds = depTable[s];
    \forall bk \in bkwds\{
      New = \text{Rewind}(bk,GState,s);
      \text{if } New \text{ = done}
        \text{break;}
      \text{else}
        \text{Backward}(New);
    \}
  \}
  \text{if all states are done}
    \text{return reached}
  \text{else}
    \text{return unreachable}
\}

The ‘Rewind’ procedure takes the global state one step backward by applying the reverse transition rules. ‘replace(s, st, GState)’ replaces \( s \) in \( GState \) with \( st \) and returns the new global state. Depending on the stimulus type of the backward rule \( bk \), different states in \( GState \) are rolled back. For \( orig \) and \( dst \) only the originator and destination of the stimulus is rolled back, respectively. For \( mcast \), all affected states are rolled back except the originator. \( mcastDownstream \) is similar to \( mcast \) except that all downstream routers or states are rolled back, while only one upstream router (the destination) is rolled back.

\[\text{Rewind}(bk,GState,s)\{
\]
  \text{if } bk \in I.S.
    \text{return done;}
  stim = stimulus(bk);
  st = start(bk);
if type(stim) = orig {
    New = replace(s, st, GState);
    return New;
}
∀cond ∈ preconds[stim] &
while src not found {
    str = start(cond);
    if str ∈ GState
        src found
}
if src not found
    return backTrack;
if type(stim) = dst {
    New = replace(s, st, GState);
    if checkMinTopo(New, stim)
        return New;
    else
        return backTrack;
if not checkConsistency(stim, GState)
    return backTrack;
New = GState;
if type(stim) = mcast
    ∀cond ∈ conds[stim]
        if end(cond) ∈ GState & not src
            New = replace(end, start, GState);
if type(stim) = mcastDownstream
    ∀cond ∈ conds[stim]
        if end(cond) ∈ GState & not upstream
            New = replace(end, start, GState);
        else if end ∈ GState & upstream
            New = replace(end, start, GState) once;
        if checkMinTopo(New, stim)
            return New;
        else
            return backTrack;
The following procedure checks for consistency of applying $stim$ to $GState$.

```plaintext
checkConsistency(stim,GState) {
  $\forall cond \in conds[stim]$ & cond has transition
  if start(cond) $\in GState$
    return False;
  else
    return True;
}
```

The following procedure checks if $GState$ contains the necessary components to trigger the stimulus.

```plaintext
checkMinTopo(GState,stim) {
  if $\exists MinTopos[stim] \subseteq GState$
    return True;
  else
    return False;
}
```

## D.5 Experimental statistics for PIM-DM

To investigate the utility of FOTG as a verification tool we ran this set of simulations. This is not, however, how FOTG is used to study protocol robustness.

We also wanted to study the effect of unreachable states on the complexity of the verification. The simulations for our case study show that unreachable states do not contribute in a significant manner to the complexity of the backward search for larger topologies. Hence, in order to use FOTG as a verification tool, it is not sufficient to add the reachability detection capability to FOTG.

The backward search was applied to the equivalent error states (for LANs with 2 to 5 routers connected). The simulation setup involved a call to a procedure similar to ‘EquivInit’ in Section C.2, with the parameter $S$ as the set of state symbols, and after an error check was done a call is made to the ‘Backward’ procedure instead of ‘ExpandSpace’.

States were classified as reachable or unreachable. For the four topologies studied (LANs with 2 to 5 routers) statistics were measured (e.g., max, min, median, average, and total) for number of calls to the ‘Backward’ and ‘Rewind’ procedures, and the number of backTracks were measured.
### Figure D.1: Simulation statistics for backward algorithms

As shown in figure D.1, the statistics show that, as the topology grows, all the numbers for the reachable states get significantly larger than those for the unreachable states (as in figure D.2), despite the fact that that the percentage of unreachable states increases with the topology as in figure D.3. The reason for such behavior is due to the fact that when the state is unreachable the algorithm reaches a dead-end relatively early (by exhausting one branch of the search tree). However, for reachable states, the algorithm keeps on searching until it reaches an initial global state. Hence the reachable states search constitutes the major component that contributes to the complexity of the algorithm.

### D.6 Results

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

For the following analyzed messages, we present the steps for topology synthesis, forward and backward implication.
Join: Following are the resulting steps for join loss:

**Synthesizing the Global State**

1. set the inspected message to Join
2. the startState of the post-condition is $F_{\text{Del},i} \implies G_I = \{F_{\text{Del},i}\}$
3. the state of the pre-condition is $NH_i \implies G_I = \{NH_i, F_{\text{Del},i}\}$
4. the stimulus of the pre-condition is Prune. Set the inspected message to Prune
5. the startState of the post-condition is $F_j$ which can be implied from $F_{\text{Del},i}$ in $G_I$
6. the state of the pre-condition is $NC_k \implies G_I = \{NH_i, F_{\text{Del},i}, NC_k\}$
7. the stimulus of the pre-condition is L. Set the inspected message to L
8. the startState of the post-condition is $NH$ which can be implied from $NC$ in $G_I$
9. the state of the pre-condition is Ext, an external event

**Forward implication**

without loss: $G_I = \{NH_i, F_{\text{Del},i}, NC_k\} \xrightarrow{\text{Join}} G_{I+1} = \{NH_i, F_j, NC_k\}$ correct state

loss w.r.t. affected routers (i.e. $R_j$): $\{NH_i, F_{\text{Del},i}, NC_k\} \xrightarrow{\text{Prune}} G_{I+1} = \{NH_i, NF_j, NC_k\}$ error state

**Backward implication**

$G_I = \{NH_i, F_{\text{Del},i}, NC_k\} \xleftarrow{\text{Prune}} G_{I-1} = \{NH_i, F_j, NC_k\} \xleftarrow{\text{PM}} G_{I-2} = \{M_i, F_j, NM_k\}$
$G_{I-3} = \{M_i, EU_j, NM_k\} \xleftarrow{\text{Prune}} G_{I-4} = \{NM_i, EU_j, NM_k\} = I.S.$

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

**Assert**: Following are the resulting steps for the Assert loss:
Synthesizing the Global State

1. set the inspected message to \textit{Assert}
2. the \textit{startState} of the post-condition is \( F_j \implies G_t = \{ F_j \} \)
3. the state of the pre-condition is \( F_i \implies G_t = \{ F_i, F_j \} \)
4. the stimulus of the pre-condition is \( FPkt_j \). Set the inspected message to \( FPkt_j \)
5. the \textit{startState} of the post-condition is \( EU_i \), in the which can be implied from \( F_i \) in \( G_i \)
6. the state of the pre-condition is \( F_j \), already in \( G_j \)
7. the stimulus of the pre-condition is \( SPkt_j \). Set the inspected message to \( SPkt_j \)
8. the \textit{startState} of the post-condition is \( NF_j \), which can be implied from \( F_j \) in \( G_j \)
9. the stimulus of the pre-condition is \( Ext \), an external event

Forward Implication

\[ G_t = \{ F_i, F_j \} \xrightarrow{\text{Assert}} G_{t+1} = \{ F_i, NF_j \} \text{ error} \]

Backward Implication

\[ G_t = \{ F_i, F_j \} \xleftarrow{FPkt_j} G_{t-1} = \{ EU_i, F_j \} \xleftarrow{SPkt_j} G_{t-2} = \{ EU_i, EU_j \} = I.S. \]

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

\textbf{Graft:} Following are the resulting steps for the \textit{Graft} loss:
Figure D.4: A topology having a \{F_1, F_j, \ldots, F_k\} LAN

Synthesizing the Global State

1. Set the inspected message to \textit{Graft}_{\text{Rec}}
2. the start\textit{State} of the post-condition is \(NF \Rightarrow G_I = \{NF\}\)
3. the end\textit{State} of the pre-condition is \(NH_{\text{Rtx}} \Rightarrow G_I = \{NF, NH_{\text{Rtx}}\}\)
4. the stimulus of the pre-condition is \textit{Graft}_{\text{Tx}}
5. the start\textit{State} of the post-condition is \(NH\) which may be implied from \(NH_{\text{Rtx}}\) in \(G_I\)
6. the end\textit{State} of the pre-condition is \(NH\) which may be implied
7. the stimulus of the pre-condition is \(H.J\), which is \textit{Ext}; i.e. external

Forward Implication

without loss: \(G_I = \{NH, NF\} \xrightarrow{\text{Graft}_{\text{Tx}}} G_{I+1} = \{NH_{\text{Rtx}}, NF\}\)
\(\xrightarrow{\text{Graft}_{\text{Tx}}} G_{I+2} = \{NH_{\text{Rtx}}, F\} \xrightarrow{\text{Ack}} G_{I+3} = \{NH, F\}\) correct state

with loss of \textit{Graft}; i.e. the \textit{Graft}_{\text{Rec}} does not take effect: \(G_I = \{NH, NF\} \xrightarrow{\text{Graft}_{\text{Tx}}} G_{I+1} = \{NH_{\text{Rtx}}, NF\}\)
\(\xrightarrow{\text{Timer Implication}} G_{I+2} = \{NH, NF\} \xrightarrow{\text{Graft}_{\text{Tx}}} G_{I+3} = \{NH_{\text{Rtx}}, F\} \xrightarrow{\text{Ack}} G_{I+4} = \{NH_{\text{Rtx}}, F\} \xrightarrow{\text{Ack}} G_{I+5} = \{NH, F\}\) correct state

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

D.7 Interleaving events and Sequencing

A \textit{Graft} message is acknowledged by the \textit{Graft – Ack} (G\textit{Ack}) message, and if not acknowledged it is retransmitted when the retransmission timer expires. In an attempt to create an erroneous scenario, the algorithm generates sequences to clear the retransmission timer, and insert an adverse event. Since the \textit{Graft} reception causes an upstream router to become a forwarder for the LAN, the algorithm interleaves a \textit{Leave} event as an adversary event to cause that upstream router to become a non-forwarder.

To clear the retransmission timer, the algorithm inserts the transition \(NH \xrightarrow{\text{G\text{Ack}}} NH_{\text{Rtx}}\) in the event sequence.
**Forward Implication**

\[ G_I = \{NH, NF\} \xrightarrow{Gra_{Fte}} G_{I+1} = \{NH_{Rtx}, NF\} \xrightarrow{GAck} G_{I+2} = \{NH, NF\} \text{ error state.} \]

**Backward Implication**

Using backward implication, we can construct a sequence of events leading to conditions sufficient to trigger the \( GAck \). From the transition table these conditions are \( \{NH_{Rtx}, F\} \):

\[ G_I = \{NH, NF\} \xleftarrow{HJ} G_{I-1} = \{NC, NF\} \xleftarrow{Del} G_{I-2} = \{NC, F_{Del}\} \xleftarrow{P_{Prune}} G_{I-3} = \{NC, F\} \xrightarrow{L} G_{I-4} = \{NH_{Rtx}, F\}. \]

To generate the \( GAck \) we continue the backward implication and attempt to reach an initial state:

\[ G_{I-4} = \{NH_{Rtx}, F\} \xrightarrow{Gra_{Fte}} G_{I-5} = \{NH_{Rtx}, NF\} \xrightarrow{Gra_{Fte}} G_{I-6} = \{NH, NF\} \xleftarrow{HJ} G_{I-7} = \{NC, NF\} \xleftarrow{Del} G_{I-8} = \{NC, F_{Del}\} \xleftarrow{P_{Prune}} G_{I-9} = \{NC, F\} \xrightarrow{P_{Fpk}} G_{I-10} = \{NM, F\} \xrightarrow{S_{Pkt}} G_{I-11} = \{NM, EU\} = I.S. \]

Hence, when a \( Graft \) followed by a \( Prune \) is interleaved with the \( Graft \) loss, the retransmission timer is reset with the receipt of the \( GAck \) for the first \( Graft \), and the systems ends up in an error state.

---

2We do not show all branching or backtracking steps for simplicity.

---

Figure D.5: Graft event sequencing
Appendix E

End-to-End Performance Evaluation

In this appendix we present details of inequality formulation for the end-to-end performance evaluation. In addition, we present the mathematical model to solve these inequalities. We also discuss the case of multiple request rounds for the timer suppression mechanism.

E.1 Conditions and Inequalities for Overhead Analysis

Given the target event, transitions are identified as either wanted or unwanted transitions, according to the maximization or minimization objective. For maximization, wanted transitions are those that establish conditions to trigger the target event, while unwanted transitions are those that nullify these conditions, and vice versa.

Let $W$ be the wanted transition, and $t(W)$ be the time of its occurrence. Let $C$ be the condition for the wanted transition, and $t(C)$ is the time at which it is satisfied, and let $U$ be the unwanted transitions occurring at time $t(U)$.

We want to establish and maintain $C$ until $W$ occurs, i.e., in the duration $[t(C), t(W)]$. Hence, $U$ may only occur outside (before or after) that interval. In Figure E.1 this means that $U$ can only occur in region 1 or region 3.

Hence, the inequalities must satisfy the following

1. the condition for the wanted transition, $C$, must be established before the event for the wanted transition, $W$, triggers, i.e., $t(C) < t(W)$, and

2. one of the following two conditions must be satisfied:

   (a) the unwanted transition, $U$, must occur before $C$, i.e., $t(U) < t(C)$, or
   (b) the unwanted transition, $U$, must occur after the wanted transition, $W$, i.e., $t(W) < t(U)$. 

132
These conditions must be satisfied for all systems. In addition, the algorithm needs to verify, using backward search and implication rules, that no contradiction exists between the above conditions and the nature of the events of the given problem.

E.1.1 Worst-case Overhead Analysis

The target event for the overhead analysis is \( p_t \).

The objective for the worst case analysis is to maximize the number of responses \( p_t \). The wanted transition is transition \( res_{\text{tmr}} [Res.(D_T \rightarrow D),p_t] \) (see Section 8.4). Hence \( t(W) = t(p_t) \). The condition for the wanted transition is \( D_T \) and its time (from transition \( tz_{\text{req}} [q_r.(D \rightarrow D_T)] \)) is \( t(C) = t(q_r) \).

The unwanted transition is one that nullifies the condition \( D_T \). Transition \( rcv_{\text{res}} [p_r.(D_T \rightarrow D)] \) is identified by the algorithm as the unwanted transition, hence \( t(U) = t(p_r) \).

For a given system \( i \), the inequalities become:

\[
t(q_{ri}) < t(p_{ti}),
\]

and either

\[
t(p_{rij}) < t(q_{ri})
\]

or

\[
t(p_{ti}) < t(p_{rij}).
\]
But from the timer expiration implication rule we get that the response time must have been set earlier by the request reception, i.e.,

\[ Res_i(\overline{D_t} \leftarrow D_{T_i}), p_t \iff q_{r_i}(\overline{D_{T_i}} \leftarrow D_i) \]

and \( t(p_t) = t(q_{r_i}) + Exp_i \). Hence,

\( t(q_{r_i}) < t(p_t) \) is readily satisfied and we need not add any constraints on the expiration timers or delays to satisfy this condition.

Thus, the inequalities formulated by the algorithm to produce worst-case behavior are:

\[ t(p_{r_{i,j}}) < t(q_{r_i}), \]

or

\[ t(p_t) < t(p_{r_{i,j}}). \]

E.1.2 Best-case Analysis

Using a similar approach to the above analysis, the algorithm identifies transition \( res_{-res}[p_r.(\overline{D_T} \to D)] \) as the wanted transition. Hence \( t(W) = t(p_r) \), and \( t(C) = t(q_r) \). The unwanted transition is transition \( res_{-tmr} \), and \( t(U) = t(p_t) \).

For system \( i \) the inequalities become:

\[ t(q_{r_i}) < t(p_{r_{i,j}}), \]

and either

\[ t(p_t) < t(q_{r_i}) \]

or

\[ t(p_{r_{i,j}}) < t(p_t). \]

But from the backward implication we have \( t(q_{r_i}) < t(p_t) \). Hence, the algorithm encounters contradiction and the inequality \( t(p_t) < t(q_{r_i}) \) cannot be satisfied.

Thus, the inequalities formulated by the algorithm to produce worst-case behavior are:

\[ t(q_{r_i}) < t(p_{r_{i,j}}), \]

and

\[ t(p_{r_{i,j}}) < t(p_t). \]
E.2 Mathematical Model for Solving the System of Inequalities

In this section we present the general model of the constraints (or inequalities) generated by our method. As a first step, we form a linear programming problem and attempt to find a solution. If a solution is not found, then we form a mixed non-linear programming problem to get the maximum number of feasible constraints.

In general, the system of inequalities generated by our method to obtain worst or best case scenarios, can be formulated as a linear programming problem.

In our case, satisfying all the constraints regardless of the objective function, leads to obtaining the absolute worst/best case. For example, in the case of worst case overhead analysis, this means obtaining the scenario leading to no-suppression.

The formulated inequalities by our method as given in Section 8.5 are as follows.

- for the worst case behavior:

\[ d_{Q,i} + Exp_i < d_{Q,j} + Exp_j + d_{j,i} , \]

or

\[ d_{Q,i} > d_{Q,j} + Exp_j + d_{j,i} . \]

- for the best case behavior:

\[ d_{Q,i} + Exp_i > d_{Q,j} + Exp_j + d_{j,i} , \]

and

\[ d_{Q,i} < d_{Q,j} + Exp_j + d_{j,i} . \]

The above systems of inequalities can be nicely represented by a linear programming model. The general form of a linear programming (LP) problem is:

\[
\text{Maximize } Z = C^T X = \sum_{0 \leq i \leq n} c_i \cdot x_i \\
\text{subject to:}
\]

\[ AX \leq B \]

\[ X \geq 0 \]
where \( Z \) is the objective function, \( C \) is a vector of \( n \) constants \( c_i \), \( X \) is a vector of \( n \) variables \( x_i \), \( A \) is \( m \times n \) matrix, and \( B \) is a vector of \( m \) elements.

The above problem can be solved practically in polynomial time using Karmarkar [Kar84] or simplex method [Dan87], if a feasible solution exists.

In some cases, however, the absolute worst/best case may not be attainable, and it may not be possible to find a feasible solution to the above problem. In such cases we want to obtain the maximum feasible set of constraints in order to get the worst/best case scenario. To achieve this, we define the problem as follows:

\[
\text{Maximize } \sum_{0 \leq i \leq m} y_i
\]

subject to:

\[
y_i \cdot f_i(x) \leq 0, \forall i
\]

\[
y_i \in \{0, 1\}
\]

or

\[
y_i \cdot (1 - y_i) = 0
\]

where \( f_i(x) \) is the original constraint from the previous problem.

This problem is a mixed integer non-linear programming (MINLP) problem, that can be solved using branch and bound methods [BM94].

E.3 Multiple request rounds

In Section 8.5 we conducted the protocol overhead analysis with the assumption that recovery will occur in one round of request. In general, however, loss recovery may require multiple rounds of request, and we need to consider the request timer as well as the response timers. Considering multiple timers or stimuli adds to the branching factor of the search. Some of these branches may not satisfy the timing and delay constraints. It would be more efficient then to incorporate timing semantics into the search technique to prune off infeasible branches.

Let us consider forward search first. For example, consider the global state \( q_{t_i}, R_{t_i} \) having a transmitted request message and a request timer running. Depending on the timer expiration value \( \text{Exp}_i \) and the delay experienced by the message \( d_{i,j} \), we may get different successor states. If \( d_{i,j} > \text{Exp}_i \) then the request timer fires first triggering the
event $Req_i$ and we get $q_{t_i}.Req_i$ as the successor state. Otherwise, the request message will be received first, and the successor state will be $q_{r_j}.R_{T_i}$. Note that in this case the timer value must be decreased by $d_{i,j}$, and taken into consideration for further forward steps. This is illustrated in figure E.2. The condition for branching is given on the arrow of the branch, and the timer value of $i$ is given by $T_i$.

![Diagram](image)

**Figure E.2: Forward search for multiple simultaneous events**

For backward search, instead of decreasing timer values (as is done with forward search), timer values are increased, and the starting point of the search is arbitrary in time, as opposed to time ‘0’ for forward search.

To illustrate, consider the global state having $(D_i \leftarrow D_{T_i}).R_{T_i}$, with the request timer running at $j$ and the response timer firing at $i$.

Figure E.3, shows the backward branching search, with the timer values at each step and the condition for each branch. In the first state, the timer $T_Q$ starts at an arbitrary point in time $x$, and the timer $T_i$ is set to ‘0’ (i.e. the timer expired triggering a response $p_{t_i}$). One step backward, either the timer at $i$ must have been started ‘$Exp_Q - x$’ units in the past, or the response timer must have been started ‘$Exp_i$’ units in the past. Depending on the relative values of these times some branch(ES) become valid. The timer values at each step are updated accordingly. Note that if a timer expires while a message is in flight (i.e. transmitted but not yet received), we use the $m$ subscript to denote it is still multicast, as in $q_{r_m}$ in the figure.

Sometimes, the values of the timers and the delays are given as ranges or intervals. Following we present how branching decision are made when comparing intervals.
Branching decision for intervals

In order to conduct the search for multiple stimuli, we need to check the constraints for each branch. To decide on the branches valid for search, we compare values of timers and delays. These values are often given as intervals, e.g., \([a, b]\).

Comparison of two intervals \(Int_1 = [a_1, b_1]\) and \(Int_2 = [a_2, b_2]\) is done according to the following rules.

Branch \(Int_1 > Int_2\) becomes valid if there exists a value in \([a_1, b_1]\) that is greater than a value in \([a_2, b_2]\), i.e., if there is overlap of more than one number between the intervals. We define the ‘<’ and ‘=’ relations similarly, i.e., if there are any numbers in the interval that satisfy the relation then the branch becomes valid.

For example, if we have the following branch conditions: (i) \(Exp_i < Exp_j\), (ii) \(Exp_i = Exp_j\), and (iii) \(Exp_i > Exp_j\). If \(Exp_i = [3, 5]\) and \(Exp_j = [4, 6]\), then, according to our above definitions, all the branch conditions are valid. However, if \(Exp_i = [3, 5]\) and \(Exp_j = [5, 7]\), then only branches (i) and (ii) are valid.

The above definitions are sufficient to cover the forward search branching. However, for backward search branching, we may have an arbitrary value \(x\) as noted above.

For example, take the state \((D_i \leftarrow D_{T_i}).R_{TQ}\). Consider the timer at \(Q\), the expiration duration of which is \(Exp_Q\) and the value of which is \(x\), and the timer at \(i\), the expiration duration of which is \(Exp_i\) and the value of which is ‘0’, as given in figure E.3. Depending on the relevant values of \(Exp_i\) and \(Exp_Q - x\), the search follows some branch(es). If \(Exp_Q = [a_1, b_1]\), then \(x = [0, b_1]\) and \(Exp_Q - x = [0, b_1]\). Hence, we can apply the
forward branching rules described earlier by taking $Exp_Q - x = [0, b_1]$, as follows. Since $Exp_i = [a_2, b_2]$, where $a_2 > 0$ and $b_2 > 0$, hence, the branch condition $Exp_i > Exp_Q - x$ is always true. The condition $Exp_i = Exp_Q - x$ is valid when: (i) $Exp_i = Exp_Q$, or (ii) $Exp_i < Exp_Q$. The last condition, $Exp_i < Exp_Q - x$, is valid only if $Exp_i < Exp_Q$.

These rules are integrated into the search algorithm for our method to deal with multiple stimuli and timers simultaneously.
Reference List


