The STRESS Method for Boundary-Point Performance Analysis of End-to-End Multicast Timer-Suppression Mechanisms

Ahmed Helmy, Sandeep Gupta, and Deborah Estrin

***Abstract*—The advent of multicast and the growth and complexity of the Internet has complicated network protocol design and evaluation. Evaluation of Internet protocols usually uses random scenarios or scenarios based on designers' intuition. Such approach may be useful for average case analysis but does not cover *boundary-point* (worst or best case) scenarios. To synthesize boundary-point scenarios, a more systematic approach is needed.**

In this paper, we present a method for automatic synthesis of worst and best case scenarios for protocol boundary-point evaluation. Our method uses a fault-oriented test generation (FOTG) algorithm for searching the protocol and system state space to synthesize these scenarios. The algorithm is based on a global finite state machine (FSM) model. We extend the algorithm with timing semantics to handle end-to-end delays and address performance criteria. We introduce the notion of a *virtual LAN* to represent delays of the underlying multicast distribution tree. Our algorithms utilize implicit backward search using branch and bound techniques and start from given *target events*.

As a case study, we use our method to evaluate variants of the timer suppression mechanism, used in various multicast protocols, with respect to two performance criteria: overhead of response messages and response time. Simulation results for reliable multicast protocols show that our method provides a scalable way for synthesizing worst case scenarios automatically. Results obtained using stress scenarios differ dramatically from those obtained through average case analyses. We hope for our method to serve as a model for applying systematic evaluation to other multicast protocols.

I. Introduction

THE recent growth of the Internet and its increased heterogeneity has introduced new failure modes and added complexity to protocol design and testing. In addition, the advent of multicast applications has introduced new challenges of qualitatively different nature than the traditional point-to-point protocols. As more complex multicast applications and protocols are coming to life, the need for systematic and automatic methods to study and evaluate such protocols is becoming more apparent. Most current approaches for protocol evaluation use average case analysis and are based on random or intuitive scenarios. Such approaches do not address protocol robustness or *boundary-point* analysis, in which the protocol exhibits worst or best case behavior. We believe that such protocol breaking points should be identified and studied in depth to understand and hopefully increase protocol robustness. At the same time we do not expect that complex adaptive protocols will be automatically verifiable under their full range of conditions. Rather, we are proposing a framework in which a protocol designer can follow a set of systematic steps, assisted by automation where possible, to cover a specific part of the design and operating space. Our goal is to complement average case studies and enrich the evaluation test-suites for multicast protocols.

In our proposed framework, a protocol designer will still need to create the initial mechanisms, describe it in the form of a finite state machine, and identify the performance criteria or correctness conditions that need to be investigated. Our automated method will pick up at that point, providing algorithms that generate scenarios or test suites that stress the protocol with respect to the identified criteria. The algorithms used in our method utilize implicit backward search using branch and bound techniques and start from given *target events*. This aims to reduce the search complexity.

Through our proposed methodology, we hope to address the following key issues of protocol design and evaluation.

- *Scenario dependent evaluation, and the use of validation test suites*: In many evaluation studies of multicast protocols the results are dependent upon several factors, such as members distribution and network topology, among others. Hence, conclusions drawn from these studies depend heavily on the evaluation scenarios.

  Protocol development usually passes through iterative cycles of refinement, which requires revisiting the evaluation scenarios to ensure that no erroneous behavior has been introduced. This brings about the need for validation test suites. Constructing these test suites can be an onerous and error-prone task if performed manually. Unfortunately, little work has been done to automate the generation of such tests for multicast network protocols. In this paper, we propose a method for synthesizing test scenarios automatically for boundary-point analysis of timer-suppression mechanisms employed by numerous multicast protocols.
- *Worst case analysis of protocols*: It is difficult to design a protocol that would perform well in all environments. However, identifying breaking points that violate correctness or exhibit worst case performance behaviors of a protocol may give insight to protocol designers and help in

Manuscript received February 18, 2001; revised January 11, 2002; approved by IEEE/ACM Transactions on Networking Editor K. Calvert. This work was supported by the Defense Advanced Research Projects Agency for the STRESS project under the Next Generation Internet program.

A. Helmy and S. Gupta are with the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles, CA 90089-2562 USA (e-mail: helmy@ceng.usc.edu; sandeep@poisson.usc.edu).

D. Estrin is with the Center for Embedded Networked Sensing, University of California, Los Angeles, CA 90095-1596 USA (e-mail: destrin@cs.ucla.edu).

Digital Object Identifier 10.1109/TNET.2003.822643

evaluating design tradeoffs. In general, it is desirable to identify, early on in the protocol development cycle, scenarios under which the protocol exhibits worst or best case behavior. The method presented in this paper automates the generation of scenarios in which multicast protocols exhibit worst and best case behaviors.

- *Performance benchmarking*: New protocols may propose to refine a mechanism with respect to a particular performance metric, using for evaluation those scenarios that show performance improvement. However, without systematic evaluation, these refinement studies often (though unintentionally) overlook other scenarios that may be relevant. To alleviate such a problem we propose to integrate stress test scenarios that provide an objective benchmark for performance evaluation.

Using our scenario synthesis methodology we hope to contribute to the understanding of better performance benchmarking and the design of more robust protocols.

A. Background on Timer Suppression

The design of multicast 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 *multiresponder* problem, where multiple members of a group respond (almost) simultaneously to an event, which causes a flood of synchronized responses throughout the network (e.g., the well-known *Ack implosion* problem) leading to performance degradation.

One common technique to alleviate this problem is the *multicast damping* technique that employs a *timer suppression* mechanism (TSM). In TSM, a group member that detects loss of a data packet multicasts a request for recovery. Other group members, that receive this request and that have previously received the data packet, schedule transmission of a response. In general, randomized timers are used in scheduling the response. While a response timer is running, if a response is received from another member then the response timer is suppressed to reduce the number of responses triggered. Consequently, the response time may be delayed to allow for more suppression. Two main performance criteria used in this case are overhead of response messages and time to recover from packet loss. Depending on the relative delays between group members and the timer settings, the mechanism may exhibit different performance. In this study, we examine worst and best case behavior of the TSM in a systematic, automatic fashion.

We are not aware of any related work that attempts to achieve this goal systematically. However, we borrow from previous work on protocol verification and test generation. We believe the TSM is a good building block to analyze as our first end-to-end case study, since it is rich in multicast and timing semantics, and can be evaluated using standard performance criteria. Furthermore, TSM is employed in several multicast protocols, including the following.

1) IP-multicast protocols, e.g., PIM [1], [2] and IGMP [3], use TSM on LANs to reduce Join/Prune control overhead.
2) Reliable multicast schemes, e.g., SRM [4] and MFTP [5], use this mechanism to alleviate *Ack implosion*. Variants of the SRM timers are used in registry replication (e.g., RRM [6], [7]) and adaptive Web caching [8].
3) Multicast address allocation schemes, e.g., AAP and SDR [9], use TSM to avoid an implosion of responses during the collision detection phase.
4) Active services [10] use multicast damping to launch one service agent ‘servent’ from a pool of servers. TSM is also used in self-organizing hierarchies (SCAN [11]), and transport protocols (e.g., XTP [12] and RTP [13]). We study several variants of TSM.

The rest of the paper is organized as follows. Section II introduces the protocol and topology models. Section III outlines the main algorithm, and Section IV presents the model for TSM. Sections V and VI present analysis of overhead and response time, and Section VII presents simulation results. Related work is described in Section VIII. Future work and conclusions are included in Section IX and X. Algorithmic details, mathematical models, and example case studies are given in the Appendixes.

II. The Model

The model is a processable representation of the system under study that enables automation of our method. Our overall model consists of the protocol model, the topology model, and the fault model.

A. Protocol Model

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

1) FSM model: Every instance of the protocol, running on a single end-system, is modeled by a deterministic FSM consisting of: 1) a set of states; 2) a set of stimuli causing state transitions; and 3) a state transition function (or table) describing the state transition rules. A protocol running on an end-system i is represented by the machine $\mathcal{M}_i = (\mathcal{S}_i, \tau_i, \delta_i)$, where $\mathcal{S}_i$ is a finite set of state symbols, τ_i is the set of stimuli, and δ_i is the state transition function $\mathcal{S}_i \times \tau_i \rightarrow \mathcal{S}_i$.

2) Global FSM model: The global state is defined as the composition of individual end-system states. The behavior of a system with n end-systems may be described by $\mathcal{M}_\mathcal{G} = (\mathcal{S}_\mathcal{G}, \tau_\mathcal{G}, \delta_\mathcal{G})$, where $\mathcal{S}_\mathcal{G}$: $\mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \times \mathcal{S}_n$ is the global state space, $\tau_\mathcal{G}$: $\bigcup_{i=1}^{n} \tau_i$ is the set of stimuli, and $\delta_\mathcal{G}$ is the global state transition function $\mathcal{S}_\mathcal{G} \times \tau_\mathcal{G} \rightarrow \mathcal{S}_\mathcal{G}$.

B. Topology Model

The topology cannot be captured simply by one metric. Indeed, its dynamics may be complex to model and sometimes intractable. We model the topology at the network layer and we abstract the network using end-to-end delays. We model the delays using the delay matrix and loss patterns using 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 Fig. 1), where $d_{i,j}$ is the delay from system i to system j.[1] The VLAN model may seem like an oversimplification of the

[1]We use the term *topology synthesis* to denote the assignment of delay values which constitute the entries of the D matrix.

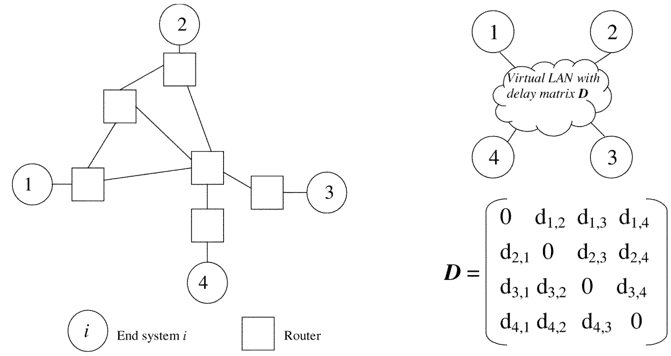


Fig. 1. Representing network details using *virtual LAN* and delay matrix $\boldsymbol{D}$. ($d_{i,j}$ is delay from i to j). (a) Detailed topology. (b) Abstract *virtual LAN*.

topology as it abstracts the internal network connectivity and queues. This, however, renders our model tractable and is quite useful in obtaining characteristics of boundary-point scenarios. We shall further investigate the utility and accuracy of our model in Section VII through detailed packet level simulations of sophisticated timer mechanisms over complex topologies.

C. Fault Model

A *fault* is a low-level (e.g., physical layer) anomalous behavior that may affect the protocol under test. Faults may include packet loss, system crashes, or routing loops. For brevity, we only consider selective packet loss in this study. Selective packet loss occurs when a multicast message is received by some group members but not others. The selective loss of a message prevents the transition that this message triggers at the intended recipient.

III. Algorithm and Objectives

To apply our method, the designer specifies the protocol as a GFSM model. In addition, the evaluation criteria, whether related to performance or correctness, are given as input to the method. In this paper, we address performance criteria; correctness has been addressed in previous studies [14], [15]. The algorithm operates on the specified model and synthesizes a set of *test scenarios*, protocol events and relations between topology delays and timer values that stress the protocol according to the evaluation criteria (e.g., exhibit maximum overhead or delay). In this section, we outline the algorithmic details of our method. The algorithm is further discussed in Section V and illustrated by a case study. Algorithmic complexity issues are discussed in Section IX.

A. Algorithm Outline

Our algorithm is a variant of the fault-oriented test generation (FOTG) algorithm presented in [15]. It includes the topology synthesis and the backward search and forward search stages. Here we 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. The algorithm is outlined in Fig. 2.

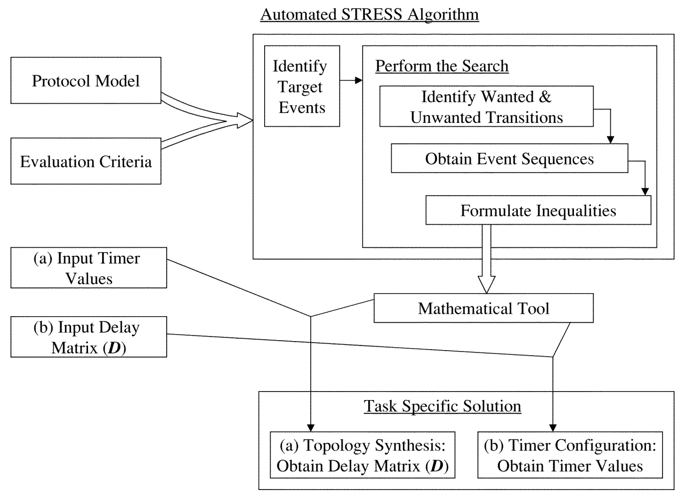


Fig. 2. STRESS framework for scenario synthesis.

1) **The target event**: The algorithm starts from a given event, called the target event. The target event (e.g., sending a message) is identified by the designer based on the protocol evaluation criteria, e.g., overhead.

2) **The search**: Three steps are taken in the search:
   a) *Identifying conditions*: The algorithm uses the protocol transition rules to identify transitions necessary to trigger the target event (called *wanted transitions*) and those that prevent it (called *unwanted transitions*).
   b) *Obtaining sequences*: Once the above transitions are identified, the algorithm uses backward and forward search to build event sequences 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. Section IV-B5 describes the implication rules.
      ii) **Forward search** is used to verify the backward search. Every backward step must correspond to valid forward step(s). Branches leading to contradictions between forward and backward search are rejected. Forward search is also used to complete event sequences necessary to maintain system consistency.
   c) *Formulating inequalities*: Based on the transitions and timed sequences obtained in the previous steps, the algorithm formulates relations between timer values and network delays that trigger the wanted transitions and avoid the unwanted transitions.

3) **Task-specific solution**: The output of the search is a set of event sequences and inequalities that satisfy the evaluation criteria. These inequalities are solved mathematically to find a topology or timer configuration, depending on the task definition.

B. Task Definition

We apply our method to two kinds of tasks.

TABLE I
STATE SYMBOL TABLE FOR THE TSM MODEL

State	Meaning
R	original state of the requester Q
R_T	requester with the request timer set
D	potential responder
D_T	responder with the response timer set

1) **Topology synthesis** is performed to identify the delays, $d_{i,j}$, in the delay matrix D that produce the best or worst case behavior, given the timer values.[2]
2) **Timer configuration** is performed to obtain the timer values that cause the best and worst case behavior, given the topology delay matrix D.

IV. TIMER SUPPRESSION MECHANISM

In this section, we present a simple description of the TSM, then present its model, used thereafter in the analysis. TSM 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 (e.g., randomly), the expiration of which, after duration Exp_i, triggers a response p. If the system i receives a response p from another system j while its timer is running, it suppresses its own response.

A. Performance Evaluation Criteria

We use two performance criteria to evaluate TSM:

1) Overhead of response messages, where the worst case produces the maximum number of responses per data packet loss. As an extreme case, this occurs when all potential responders respond and no suppression takes place.
2) The response delay, where worst case scenario produces maximum loss recovery time.

B. Timer Suppression Model

Following is the TSM model used in the analysis.

1) Protocol States ($\mathcal{S}$)*:* The state symbol table for the TSM model is given in Table I.

2) Stimuli or Events:

1) Sending/receiving messages: transmitting response (p_t) and request (q_t), receiving response (p_r) and request (q_r).
2) Timer events and other events: the events of firing the request timer, Req, and response timer, Res, and the event of detecting packet loss, L.

3) Notation: Following are the notations used in the transition table and the analysis thereafter.

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 scheduled reception 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., $p_{r_{i,j}}$ denotes system i receiving response from system j.

[2]If the topology connectivity is also given, the task may also include obtaining *link* delays, as discussed in Appendix B.

TABLE II
TRANSITION TABLE FOR TSM

Symbol	Event	Effect	Meaning
$loss$	L	$(R \rightarrow R_T).q_t$	Loss detection causes q_t and setting of request timer
tx_req	q_t	q_{r_m}	Transmission of q causes multicast reception of q after network delay
rcv_req	q_r	$D \rightarrow D_T$	Reception of q causes a system in D state to set response timer
res_tmr	Res	$(D_T \rightarrow D).p_t$	Response timer expiration causes transmission of p & a change to D
tx_res	p_t	p_{r_m}	Transmission of p causes multicast reception of p after network delay
rcv_res	p_r	$R_T \rightarrow R$, $D_T \rightarrow D$	Reception of p by a system with the timer set causes suppression
req_tmr	Req	q_t	Expiration of request timer causes re-transmission of q

The state subscript T denotes 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 Exp.

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

Effect in the transition table may contain transition and stimulus in the form $(startState \rightarrow endState)$*.stimulus*, which indicates that the condition for triggering *stimulus* is the state transition. An effect may contain several transitions (e.g., *Trans*1, *Trans*2), which means that out of these transitions only those with satisfied conditions will take effect.

To describe event sequences in the backward search we denote $a \Leftarrow b$, where a and b are global states, which means that a succeeds b in the event sequence, and that b can be inferred from a. Also, for forward search we use $a \Rightarrow b$, which means that a precedes b in the event sequence and that a implies b.

4) Transition Table: The transition table for TSM is given in Table II.

The model contains a requester, Q, and several potential responders (e.g., i and j).[3] Initially, the requester, Q, exists in state R and all potential responders exist in state D. Let t_0 be the time at which Q sends the request, q. 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., q_{r_j} occurs at $t(q_{r_j})$.[4]

5) Implication Rules: The backward search uses the following cause-effect implication rules:

1) Transmission/Reception **(Tx_Rcv)**: By the reception of a message, the algorithm infers the transmission of that message—without loss—sometime in the past (after applying the network delays). An example of this implication is $p_{r_{i,j}} \Leftarrow p_{t_j}$, where $t(p_{r_{i,j}}) = t(p_{t_j}) + d_{j,i}$.
2) Timer Expiration **(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

[3]Since there is only one requester, we simply use q_t instead of q_{t_Q}, q_{r_i} instead of $q_{r_{i,Q}}$, Req instead of Req_Q, R instead of R_Q and R_T instead of R_{T_Q}.

[4]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.

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.[5]

3) State Creation **(St_Cr)**: To build a history of events leading to a certain state, we reverse the transition rules and get to the $startState$ of the transitions leading to the creation of the state in question. For example, $D_{T_i} \Leftarrow (D_{T_i} \leftarrow D_i)$ means that for the system to be in state D_{T_i} the system must have existed in state D_i and this is inferred from the transition $(D_{T_i} \leftarrow D_i)$.

In the following sections, we use the above model to synthesize worst and best case scenarios according to protocol overhead and response time.

V. Protocol Overhead Analysis

In this section, we conduct worst and best case performance analyses for TSM with respect to the number of responses triggered per packet loss. Initially, we assume no loss of request or response messages until recovery, and that the request timer is high enough that the recovery will occur within one request round. The case of multiple request rounds is discussed in Appendix C.

A. Worst Case Analysis

Worst case analysis aims to obtain scenarios with maximum number of responses per data loss. In this section, we present the algorithm to obtain inequalities that lead to worst case scenarios. These inequalities are a function of network delays and timer expiration values.

1) Target Event: 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.

2) The Search: As previously described in Section III-A, the main steps of the search algorithm are: 1) to identify the wanted and unwanted transitions; 2) to obtain sequences leading to the above transitions, and calculating the times for these sequences; and 3) to formulate 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. The algorithm also 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). From the transition table described in Section IV-B4, the algorithm identifies transition res_tmr, or $Res.(D_T \rightarrow D).p_t$, as a *wanted transition* and its condition D_T as a *wanted condition*. Transition rcv_req, or $q_r.(D \rightarrow D_T)$, is also identified as a *wanted transition* since it is necessary to create D_T. The *unwanted transition* is identified as transition rcv_res, or $p_r.(D_T \rightarrow D)$, 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) wanted transition, res_tmr; 2) wanted transition rcv_req; and 3) unwanted transition rcv_res, as follows:

  1) To obtain the sequence of events for transition res_tmr, the algorithm applies implication rules (see Section IV-B5) Tmr_Exp, St_Cr, Tx_Rcv in that order, and we get $res\_tmr_i \Leftarrow rcv\_req_i \Leftarrow tx\_req$, or

$$Res_i.\left(D_i \leftarrow D_{T_i}\right).p_{t_i} \Leftarrow q_{r_i}.\left(D_{T_i} \leftarrow D_i\right) \Leftarrow q_t.$$

  Hence, the calculated time for $t(p_{t_i})$ becomes

$$t\left(p_{t_i}\right) = t_0 + d_{Q,i} + Exp_i,$$

  where t_0 is the time at which q_t occurs.

  2) To obtain the sequence of events for transition rcv_req the algorithm applies implication rule Tx_Rcv, and we get $rcv\_req_i \Leftarrow tx\_req$, or

$$q_{r_i}.\left(D_{T_i} \leftarrow D_i\right) \Leftarrow q_t.$$

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

$$t\left(q_{r_i}\right) = t_0 + d_{Q,i}.$$

  3) To obtain 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 $rcv\_res_i \Leftarrow res\_tmr_j \Leftarrow rcv\_req_j \Leftarrow tx\_req$, or

$$p_{r_{i,j}}.\left(D_i \leftarrow D_{T_i}\right) \Leftarrow Res_j.\left(D_j \leftarrow D_{T_j}\right).p_{t_j} \Leftarrow q_{r_j}.\left(D_{T_j} \leftarrow D_j\right) \Leftarrow q_t.$$

  Hence, the calculated time for $t(p_{r_{i,j}})$ becomes

$$t\left(p_{r_{i,j}}\right) = t_0 + d_{Q,j} + Exp_j + d_{j,i}.$$

- **Formulating Inequalities**: Based on the above wanted and unwanted transitions, the algorithm forms constraints and conditions to aoivd the unwanted transition, rcv_res, while invoking the wanted transition, res_tmr, to transit out of D_T. To achieve this, the algorithm automatically derives the following inequality (see Appendix A for more details):

$$t\left(p_{t_i}\right) < t\left(p_{r_{i,j}}\right). \tag{1}$$

  Substituting expressions for $t(p_{t_i})$ and $t(p_{r_{i,j}})$ previously derived, we get

$$d_{Q,i} + Exp_i < d_{Q,j} + Exp_j + d_{j,i}.$$

  Alternatively, we can avoid the unwanted transition rcv_res if the system did not exist in D_T when the response is received. Hence, the algorithm automatically derives the following inequality (see Appendix A for more details):

$$t\left(p_{r_{i,j}}\right) < t\left(q_{r_i}\right). \tag{2}$$

[5]We use the notation $Event.Effect$ to represent a transition.

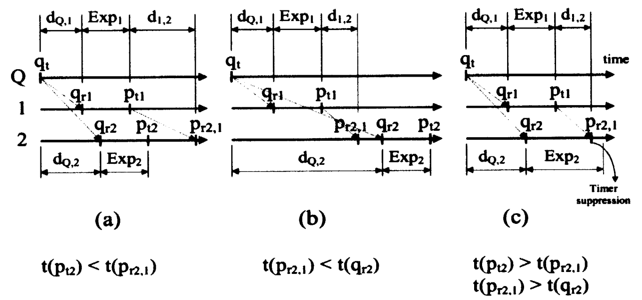


Fig. 3. Event sequencing: sequences (a) and (b) do not lead to suppression, whereas (c) does lead to suppression.

Again, substituting expressions derived above, we get

$$d_{Q,i} > d_{Q,j} + Exp_j + d_{j,i}.$$

Note that (1) and (2) are general for any number of responders, where i and j are any two responders in the system. Fig. 3(a) and (b) shows (1) and (2), respectively.

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 assigning positive values to the delays $d_{i,j}\,\forall i, j$.

  In (1), if we take $d_{Q,i} = d_{Q,j}$,[6] we get

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

  These inequalities put a lower 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 and unicast routing protocols, tree type and dynamics, propagation, transmission and queueing delays. One simple topology that reflects the delays of the delay matrix is a completely connected network where the underlying multicast distribution tree coincides with the unicast routing. There may also exist many other complex topologies that satisfy the delay matrix D.
- **Timer configuration**: Given the delay values, ranges or bounds, we want to obtain timer expiration values that produce worst case behavior. We obtain a range for the relative timer settings (i.e., $Exp_i - Exp_j$) using (1).

  The solution for the system of inequalities given by (1) and (2) can be solved in the general case using linear programming (LP) techniques (see Appendix B for more details). Section VII uses the above solutions to synthesize simulation scenarios. Note, however, that it may not be feasible to satisfy all these 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 packet loss. This problem is further discussed in Appendix B.

B. Best Case Analysis

Best case overhead analysis constructs constraints that lead to maximum suppression, i.e., minimum number of responses. The following conditions are formulated using steps similar to those given in the worst case analysis:

$$t\left(p_{t_i}\right) > t\left(p_{r_{i,j}}\right) \tag{3}$$

and

$$t\left(p_{r_{i,j}}\right) > t\left(q_{r_i}\right). \tag{4}$$

These are complementary conditions to those given in the worst case analysis. Fig. 3(c) shows (3) and (4). Refer to Appendix A for more details on the inequality derivation.

This concludes our description of the algorithmic details to construct worst and best case delay-timer relations for overhead of response messages. Solutions to these relations represent delay and timer settings for stress scenarios that are used later on for simulations.

VI. Response Time Analysis

In this section, we conduct the performance analysis with respect to response time, i.e., the time for the requester to recover from the packet loss. 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.

To systematically approach this problem, we consider the following three cases.

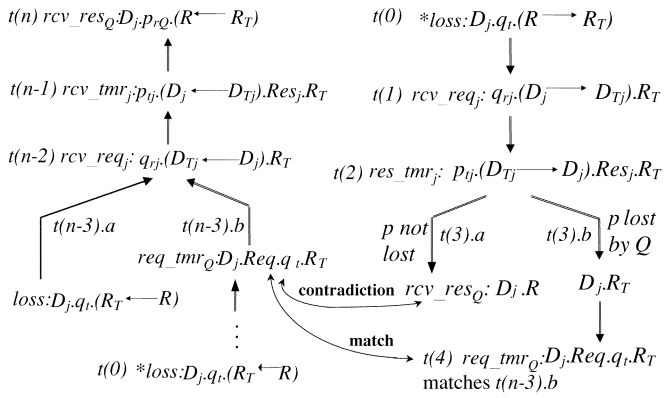
1) The case of *no* loss to the response message. This case leads to single round of request-response messages. Without loss of response messages this problem becomes one of maximizing the round trip delay between the requester and the first responder.
2) The case of single selective loss of the response message. In selective loss the response may be received by some systems but not others. This case may lead to two rounds of request-response messages. We analyze this case in the first part of this section.
3) The case of multiple selective losses of the response messages. This case may lead to more than two rounds of request-response messages, and is discussed at the end of this section.

We now consider the case of **single selective loss** of the response message during the recovery phase. For selective losses, transition rules are applied to only those systems that receive the message.

A. 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 interval is $t(p_{r_Q}) - t(q_t) = t(p_{r_Q}) - t_0$. The designer identifies $t(p_{r_Q})$ as the target time, hence, p_{r_Q} is the target event.

[6]The number of inequalities (n^2, where n is the number of responders) is less then the number of the unknowns $d_{i,j}(n^2 - n)$, hence, there are multiple solutions. We can obtain a solution by assigning values to n unknowns (e.g., $d_{Q,i}$) and solving for the others.



(a) (b)

Fig. 4. Backward and forward search for response time analysis. (a) Backward search starts from the loss recovery state (at time $t(n)$) when the requester receives a response (rcv_res_Q), and ends at the initial (packet *loss*) state at time $t(0)$. Part of the event sequence is incomplete (denoted by *dots*). (b) Forward search starts (at time $t(0)$) from the last state reached by the backward search after the *dots* (denoted by $*loss$). It attempts to fill the gap in the sequence while checking for consistency, and finds a match.

B. The Search

We present in detail the case of single responder, then discuss the multiple responders case.

- **Backward search:** As shown in Fig. 4(a), the backward search starts from p_{R_Q} and is performed over the transition table (see Section IV-B4) using the implication rules in Section IV-B5, yielding $rcv\_res_Q \Leftarrow res\_tmr_j \Leftarrow rcv\_req_j$, or[7]

$$D_j.p_{r_Q}.(R \leftarrow R_T) \Leftarrow p_{t_j}.\left(D_j \leftarrow D_{T_j}\right).Res_j.R_T$$
$$\Leftarrow q_{r_j}.\left(D_{T_j} \leftarrow D_j\right).R_T$$

  at which point the algorithm reaches a branching point, where two possible preceding states could cause q_{r_j}:
  1) The first is transition *loss*, or $D_j.q_t.(R_T \leftarrow R)$, and since the initial state R is reached, the backward search ends for this branch.
  2) The second is transition req_tmr, or $D_j.Req.q_t.R_T$. Note that Req indicates the need for a transition to R_T, i.e., $(R_T \leftarrow R)$, and the search for this last state yields the intial (data packet loss) state *loss*: $D_j.q_t.(R_T \leftarrow R)$. However, q_t is message transmission, which implies that the message must be received (or lost). Hence, there are gaps in the event sequence [indicated by the *dots* in Fig. 4(a)] that are filled through forward search [in Fig. 4(b)].
- **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, as shown in Fig. 4(b), one possible forward sequence from the initial state gives $loss \Rightarrow tx\_req \Rightarrow rcv\_req_j \Rightarrow res\_tmr_j \Rightarrow tx\_res_j$, or

$$D_j.q_t.(R \rightarrow R_T) \Rightarrow q_{r_j}.\left(D_j \rightarrow D_{T_j}\right).R_T$$
$$\Rightarrow p_{t_j}.\left(D_{T_j} \rightarrow D_j\right).Res_j.R_T.$$

  The algorithm then searches two possible next states:
  1) If p_{t_j} is not lost, and hence, causes p_{r_Q}, then the next state is $D_j.R$. But the original backward search started from $D_j.q_t.Req.R_T$, which cannot be reached from $D_j.R$. Hence, we get contradiction and the algorithm rejects this sequence.
  2) If the response p is lost by Q, we get $D_j.R_T$ that leads to $D_j.Req.q_t.R_T$. The algorithm identifies this as a feasible sequence.

  Calculating the time for each feasible sequence, the algorithm identifies the latter sequence as one of maximum response time.

For **multiple responders**, the algorithm automatically explores the different possible selective loss patterns of the response message. The search identifies the sequence with maximum response as one in which only one responder triggers a response that is selectively lost by the requester. To construct such a sequence, the algorithm creates conditions and inequalities similar to those formulated for the best case overhead analysis with respect to number of responses (see Section V-B).

Effectively, the sequence obtained above occurs when the response is lost by the requester, which triggers another request. Intuitively, the response delay is increased with multiple request rounds. The case of **multiple selective losses** of the response messages may trigger multiple (more than two) request rounds. Practically, the number of request rounds is bounded by the protocol implementation, which imposes an upper bound on the number of requests sent per packet loss. This, in turn, imposes an upper bound on the worst case response time. This bound can be easily integrated into the search to end the search when the maximum number of allowed request rounds is reached.[8]

After conducting the above analyses, we have applied our method to generate worst case overhead scenarios for topology synthesis and timer configuration tasks using determinsitic and adaptive timers (see Appendix D). We also applied it to response time analysis and to best case analyses. In the next section, we show network simulations using our generated worst case overhead scenarios.

C. On Algorithmic Complexity

One goal of our case studies is to understand and evaluate the computational complexity of our method and algorithms. Our main algorithm uses a mix of backward and forward search techniques. The algorithm starts from *target events* and uses implicit backward search and branch and bound techniques to synthesize the required scenario sequences. Complexity of such algorithm depends on the FSM, the state transition rules, and the target events from which the algorithm starts. Hence, it is hard to quantify, in general terms, the complexity of our algorithm. Nonetheless, we shall comment on the nature of the method and

[7]The GFSM may be represented by composition of individual states (e.g., $State_1.State_2$ or $transition_1.State_2$).

[8]The theoretical, trivial, worst case response time is an infinite number of request rounds. The goal of this analysis, however, is to provide a scenario in which response time is maximized. It was a finding of our algorithm that if multiple rounds are forced then the response time increases. It was also part our algorithm to formulate conditions under which multiple response rounds are forced.

the algorithm qualitatively based on our case studies. We note the following.

1) Our algorithms use branch and bound techniques and utilize implicit backward search starting from a target event (versus explicit forward reachability analysis starting from initial states). Branch and bound techniques are, generally, hard to quantify in terms of (worst case or average case) complexity in abstract terms. Although the worst case for branch and bound could be exponential, through our experiments we found that, on average, the target-based approach has far less complexity than forward search. In many cases the branch bounds immediately (e.g., due to contradiction if the sequence is not feasible). For all our STRESS case studies, we have found our search algorithms to be quite manageable.
2) Scenarios synthesized using the STRESS method usually are simple and include relatively small topologies. Thus, they often experience low computational complexity. It is our observation, in all our case studies thus far, that erroneous and worst case protocol behaviors may be invoked using relatively simple (yet carefully synthesized) scenarios. Also, it is often the case that these simple scenarios are extensible to larger and more complex scenarios using simple heuristics. In Section VII, we shall demonstrate how the simple scenarios generated by STRESS, with only a few receivers, could be scaled up to include hundreds of receivers. Accuracy of such extrapolation is validated through detailed simulations.

VII. Systematic STRESS Simulations

To evaluate the utility and accuracy of our method, we have conducted a set of detailed simulations for the Scalable Reliable Multicast (SRM) [4] based on our worst case scenario synthesis results for the timer-suppression mechanism. We tied our method to the network simulator (NS) [16]. The output of our method, in the form of inequalities (see Section V), is solved using a mathematical package (LINDO). The solution, in terms of a delay matrix, is then used to generate the simulation topologies for NS automatically.

For our simulations, we measured the number of responses triggered for each data packet loss. We have conducted two sets of simulations, each using two sets of topologies. The simulated topologies included topologies with up to 200 receivers. The first set of topologies was generated according to the overhead analysis presented in this paper. We call this set of topologies the *stress* topologies. Example *stress* topology is shown in Fig. 5(a), and its corresponding fully connected topology is shown in Fig. 5(b). Both topologies satisfy the delay matrix D produced by stress. The second set of topologies was generated by the GT-ITM topology generator [17], generating random and transit stub topologies.[9] We call this set of topologies the *random* topologies.

The first set of simulations was conducted for the SRM deterministic timers. SRM response timer values are selected randomly from the interval $[D_1.d_r, (D_1 + D_2).d_r]$, where d_r is the estimated distance to the requester, and D_1, D_2 depend on the timer type. For deterministic timers, $D_2 = 0$ and $D_1 = 1$. The results of the simulation are shown in Fig. 6(a). The number of responses triggered for all the *stress* topologies was $n-1$, where n is the number of receivers (i.e., no suppression occurred). For the *random* topologies, with up to 200 receivers, the number of responses triggered was less than 20 responses in the worst case.

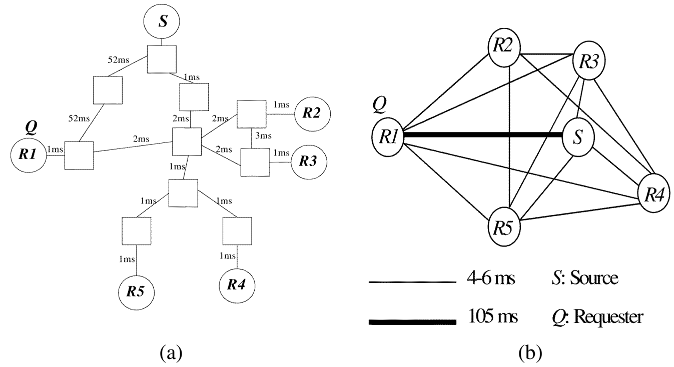


Fig. 5. Example *stress* topology used for the simulation. (a) Detailed topology. (b) Abstract end-to-end topology.

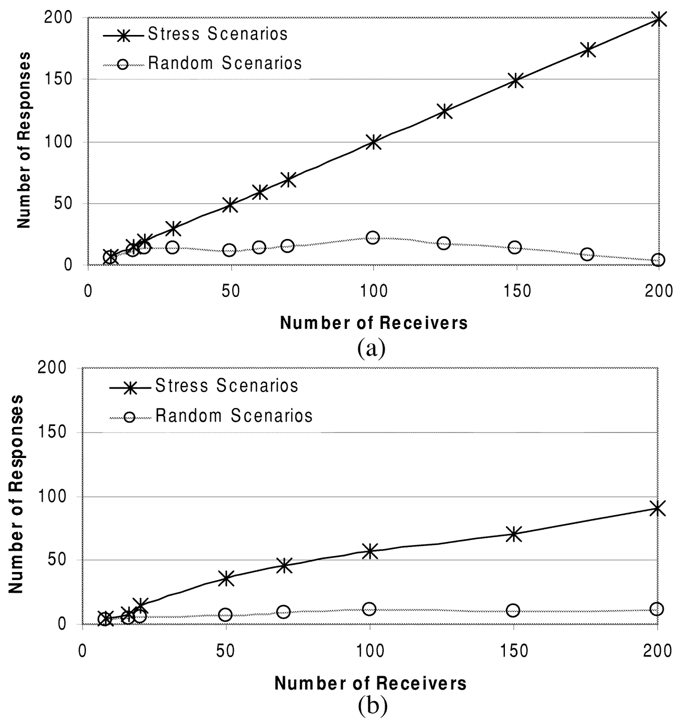


Fig. 6. Simulation results for deterministic and adaptive timers over *stress* and *random* topologies. (a) SRM with deterministic timers. (b) SRM with adaptive timers.

Using the same two sets of topologies, the second set of simulations was conducted for the SRM adaptive timers. Adaptive timers adjust their interval based on the number of duplicate responses received and the estimated distance to the requester. The results are given in Fig. 6(b). For the *stress* topologies, almost 50% of the receivers triggered responses, whereas *random* topologies simulation generated almost ten responses in the worst case, for topologies with 100–200 receivers.

These simulations illustrate how our method may be used to generate consistent worst case scenarios in a scalable fashion.

[9]This topology generator is commonly used in networking research. Using GT-ITM we covered most topologies used in several SRM studies [18], [19].

It is interesting to notice that worst case topologies generated for simple deterministic timers also experienced substantial overhead (perhaps not the worst, though) for more complicated timers (such as the adaptive timers). It is also obvious from the simulations that *stress* scenarios are more consistent than the other scenarios when used to compare different mechanisms, in this case deterministic and adaptive timers; the performance gain for adaptive timers is very clear under *stress* scenarios.

So, in addition to experiencing the worst case behavior of a mechanism, our *stress* methodology may be used to compare protocols in the above fashion and to aid in investigating design tradeoffs. It is a useful tool for generating meaningful simulation scenarios that we believe should be considered in performance evaluation of protocols in addition to the average case performance and random simulations. We plan to apply our method to test a wider range of protocols through simulation. We have conducted other case studies using our *stress* method on multicast routing (PIM-DM [15], [20], PIM-SM [14]), MARS, Mobile-IP [21], and multicast-congestion control [22], [23]. We are currently investigating ad hoc network protocols (e.g., MAC layer, ad hoc routing and geographic routing).

VIII. Related Work

Related work falls mainly in the areas of protocol verification, VLSI test generation, and network simulation.

There is a large body of literature dealing with verification of protocols. Verification systems typically address well-defined properties—such as *safety*, *liveness*, and *responsiveness* [24]—and aim to detect violations of these properties. In general, the two main approaches for protocol verification are theorem proving and reachability analysis [25]. Theorem proving systems define a set of axioms and relations to prove properties, and include *model-based* and *logic-based* formalisms [26], [27]. These systems are useful in many applications. However, these systems tend to abstract out some network dynamics that we study (e.g., selective packet loss). Moreover, they do not synthesize network topologies and do not address performance issues *per se*.

Reachability analysis algorithms [28], on the other hand, try to inspect reachable protocol states, and suffer from the "state space explosion" problem. To circumvent this problem, state reduction techniques could be used [29]. These algorithms, however, do not synthesize network topologies. Reduced reachability analysis has been used in the verification of cache coherence protocols [30], using a GFSM model. We adopt a similar FSM model and extend it for our approach in this study. However, our approach differs in that we address end-to-end protocols that encompass rich timing, delay, and loss semantics, and we address performance issues, e.g., overhead and delay.

There are many publications dealing with conformance testing [31]–[34]. However, conformance testing verifies that an implementation (as a black box) adheres to a given specification of the protocol by constructing input/output sequences. Conformance testing is useful during the implementation testing phase—which we do not address in this paper—but does not address performance issues nor topology synthesis for design testing. By contrast, our method synthesizes test scenarios for protocol design, according to evaluation criteria.

Automatic test generation techniques have been used in several fields. VLSI chip testing [35] uses test vector generation to detect target faults. Test vectors may be generated based on circuit and fault models, using the fault-oriented technique, that utilizes *implication* techniques. These techniques were adopted in [15] to develop fault-oriented test generation (FOTG) for multicast routing. In [15], FOTG was used to study correctness of a multicast routing protocol on a LAN. We extend FOTG to study performance of end-to-end multicast mechanisms. We introduce the *virtual LAN* (VLAN) concept to represent the underlying network, integrate timing and delay semantics into our model, and use performance criteria to drive our synthesis algorithm.

In [14], a simulation-based stress testing framework based on heuristics was proposed. However, that method does not provide automatic topology generation, nor does it address performance issues. The VINT [16] tools provide a framework for Internet protocols simulation. Based on the NS [16] and the network animator (NAM) [36], VINT provides a library of protocols and a set of validation test suites. However, it does not provide a generic tool for generating these tests automatically. The work in this paper is complementary to such studies, and may be integrated with network simulation tools similar to our work in Section VII.

IX. Future Work

In this paper, we have presented our first endeavor to automate the test synthesis as applies to boundary-point performance evaluation of multicast timer suppression protocols. Our case studies were by no means exhaustive. However, they gave us insights into the research issues involved. Particularly, in this section we present our future plans to explore several potential extensions and applications of our method.

Automated generation of simulation test suites: 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. The inputs to the simulation may include the topology, host events (such as traffic models), network dynamics (such as link failures or packet loss) and membership distribution and dynamics. Our future work includes 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 multicast 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 reusing established mechanisms, as appropriate, in the protocol design process. Identifying, verifying, and understanding building blocks for such mechanisms is necessary to increase their reusability. 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 this study and earlier works [14], [15], some mechanistic building blocks for multicast protocols were identified, namely, the timer-suppression mechanism and the Join/Prune mechanism (for multicast routing). More work is needed to identify more building blocks to cover a wider range of protocols.

Generalization to performance bound analysis: An approach similar to the one we have taken in this paper may be based on performance bounds, instead of worst or best case analyses. We call such approach *condition-oriented test generation*.

For example, a target event may be defined as "the response time exceeding certain delay bounds" (either absolute or parametrized bounds). 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 and analyze quality-of-service or real-time protocols, for example.

Applicability to other problem domains: So far, our method has been applied mainly to case studies on multicast protocols 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 assume dynamic topologies and lossy channels, and deal with stringent power constraints, which differentiates their protocols from Internet protocols [37]. Possible research directions in this respect include the following:

- 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;
- investigating the algorithms and search techniques that best fit the new model or evaluation criteria.

X. Conclusion

We have presented a methodology for scenario synthesis for boundary-point performance evaluation of multicast protocols. In this paper, we applied our method to worst and best case evaluation of the timer suppression mechanism; a common building block for various multicast protocols. We introduced a *virtual LAN* model to represent the underlying network topology and an extended GFSM model to represent the protocol mechanism. We adopted the fault-oriented test generation algorithm for search, and extended it to capture timing/delay semantics and performance issues for end-to-end multicast protocols.

Two performance criteria were used for evaluation of the worst and best case scenarios: the number of responses per packet loss and the response delay. Simulation results illustrate how our method can be used in a scalable fashion to test and compare reliable multicast 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.

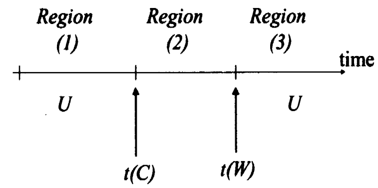


Fig. 7. Timeline for transition ordering.

Appendixes: Algorithmic Details

In these Appendixes, 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, and present several example case studies.

Appendix A

Deriving Stress Inequalities

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.

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

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 Fig. 7, this means that U can only occur in *Region*(1) or *Region*(3).

Hence, the inequalities must satisfy the following:

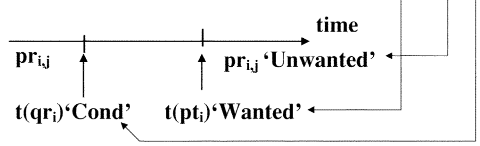
1) the condition for the wanted transition, C, must be established before the event for the wanted transition, W, triggers, i.e., $t(C) < t(W)$, and
2) one of the following two conditions must be satisfied:
   a) the unwanted transition, U, must occur before C, i.e., $t(U) < t(C)$, or
   b) the unwanted transition, U, must occur after the wanted transition, W, i.e., $t(W) < t(U)$.

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 protocol.

A. 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

Symbol	Event	Effect	Time
res_tmr	Res	$(D_{Ti} \rightarrow D_i).pt_i$	$t(pt_i)$
rcv_req	qr_i	$D_i \rightarrow D_{Ti}$	$t(qr_i)$
rcv_res	$pr_{i,j}$	$D_{Ti} \rightarrow D_i$	$t(pr_{i,j})$



t(Cond) < t(Wanted) → $t(qr_i) < t(pt_i)$
and
[t(Unwanted) < t(Cond) → $t(pr_{i,j}) < t(qr_i)$
or
t(Wanted) < t(Unwanted) → $t(pt_i) < t(pr_{i,j})$]

Fig. 8. Formulating the inequalities automatically.

responses p_t. The wanted transition is transition res_tmr, or $Res.(D_T \rightarrow D).p_t$ (see Section IV). Hence, $t(W) = t(p_t)$. The condition for the wanted transition is D_T and its time is $t(C) = t(q_r)$, from transition tx_req, or $q_r.(D \rightarrow D_T)$.

The unwanted transition is one that nullifies the condition D_T. Transition rcv_res, or $p_r.(D_T \rightarrow D)$, is identified by the algorithm as the unwanted transition, hence, $t(U) = t(p_r)$.

For a given system i, the inequalities become

$$t(q_{r_i}) < t(p_{t_i})$$

and either

$$t(p_{r_{i,j}}) < t(q_{r_i})$$

or

$$t(p_{t_i}) < t(p_{r_{i,j}}). \tag{A.1}$$

The above automated process is shown in Fig. 8. From the timer expiration implication rule, however, we get that the response time must have been set earlier by the request reception, i.e., $Res_i.(D_i \leftarrow D_{T_i}).p_{t_i} \Leftarrow q_{r_i}.(D_{T_i} \leftarrow D_i)$ and $t(p_{t_i}) = t(q_{r_i}) + Exp_i$. Hence, $t(q_{r_i}) < t(p_{t_i})$ 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

$$t(p_{t_i}) < t(p_{r_{i,j}}).$$

B. Best Case Analysis

Using a similar approach to the above analysis, the algorithm identifies transition rcv_res, or $p_r(D_T \rightarrow D)$, 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

$$t(p_{t_i}) < t(q_{r_i})$$

or

$$t(p_{r_{i,j}}) < t(p_{t_i}).$$

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

Thus, the inequalities formulated by the algorithm to produce best case behavior are

$$t(q_{r_i}) < t(p_{r_{i,j}})$$

and

$$t(p_{r_{i,j}}) < t(p_{t_i}).$$

Appendix B

Solving System of Inequalities

In this Appendix, 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 nonlinear 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 V 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 represented by a linear programming (LP) model. The general form of an LP problem is

$$\text{Maximize} Z = C^T X = \sum_{0 \le i \le n} c_i \cdot x_i$$

subject to:

$$AX \le B$$
$$X \ge 0$$

where Z is the objective function (in our case it is a dummy objective function such as $Z = \mathrm{const}$), 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. This problem can be solved practically in polynomial time using the Karmarkar [38] or simplex method [39], 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:

$$\text{Maximize} \sum_{0 \le i \le m} y_i$$

subject to:

$$y_i \cdot f_i(x) \le 0 \quad \forall i$$
$$y_i \in \{0, 1\}$$

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 nonlinear programming (MINLP) problem, that can be solved using branch and bound methods [40].

Obtaining link delays: In the previous discussion, we assumed that the model deals only with end-to-end delays ($d_{i,j}$ of the delay matrix D). In some cases, however, it may be the case that the connectivity of the network topology is given and the task is to find the *link* delays (instead of end-to-end delays). We present a very simple extension to the model to accommodate such situation, as follows. Let l_x be any link in the topology and let d_{l_x} be its delay. Take any two end systems i and j and let the path from i to j pass through links $l_a, l_b, \ldots l_n$. Hence, we get $d_{i,j} = \sum_{x \in L} d_{l_x}$, where $L = \{l_a, l_b, \ldots l_n\}$. Substituting these relations in the above inequalities we can formulate the problem in terms of link delays.

APPENDIX C
MULTIPLE REQUEST ROUNDS

In Section V, 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 state $q_{t_i}.R_{T_i}$ having a transmitted request message and a request timer running. Depending on the timer expiration value Exp_i and the delay experienced by the message $d_{i,j}$, we may get different successor states. If $d_{i,j} > Exp_i$ then the request timer fires first triggering the event Req_i and we get $q_{t_i}.Req_i$

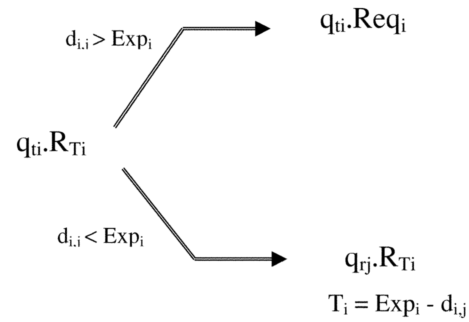


Fig. 9. Forward search for multiple simultaneous events.

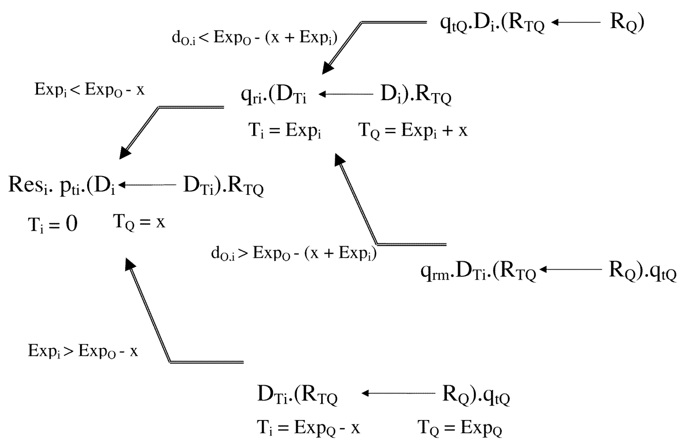


Fig. 10. Backward search for multiple simultaneous events.

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 decremented by $d_{i,j}$. This is illustrated in Fig. 9. The condition for branching is given on the arrow of the branch, and the timer value of i is given by T_i.

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 state having $(D_i \leftarrow D_{T_i}).R_{T_j}$, with the request timer running at j and the response timer firing at i.

Fig. 10 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. In the following, we present how branching decisions 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 $(D_i \leftarrow D_{T_i}).R_{T_Q}$. 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 Fig. 10. 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.

APPENDIX D

EXAMPLE CASE STUDIES

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

A. *Topology Synthesis*

In this section, 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 investigate two examples of topology synthesis, one uses timers with fixed randomization intervals and the other uses timers that are a function of distance.

Let Q be the requester and 1, 2, and 3 be potential responders. Let V_{t_i} be the time required for system i to trigger a response transmission from the time a request was sent, i.e., $V_{t_i} = d_{Q,i} + Exp_i$. From Section V, we get $V_{t_i} < V_{t_j} + d_{j,i}$ for worst case overhead.

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). 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 (1) for the worst case (see Section V) we get

$$\begin{aligned} V_{t_2} &< V_{t_1} + d_{1,2} \\ V_{t_3} &< V_{t_1} + d_{1,3} \\ V_{t_3} &< V_{t_2} + d_{2,3}. \end{aligned} \tag{5}$$

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

1) Timers With Fixed Randomization Intervals: Some multicast applications and protocols (such as wb [4], IGMP [3], or PIM [41]) employ fixed randomization intervals to set the suppression timers. For instance, for the shared white board (wb) [4], the response timer is assigned a random value from the (uniformly distributed) interval $[\mathrm{t}, 2^*\mathrm{t}]$ where $\mathrm{t} = 100$ ms for the source src, and 200 ms for other responders.

Assume Q is a receiver with a lost packet. Using wb parameters, we get $Exp_{src} = [100, 200]$ ms, and $Exp_i = [200, 400]$ ms for all other nodes.

To derive worst case topologies from inequalities (A.1), we may use a standard mathematical tool for linear or nonlinear programming; for more details, see Appendix B. However, in the following, we illustrate general techniques that may be used to obtain the solution.

From inequalities (A.1), we get

$$d_{Q,2} + Exp_2 = V_{t_2} < V_{t_1} + d_{1,2} = d_{Q,1} + Exp_1 + d_{1,2}.$$

This can be rewritten as

$$d_{Q,2} - (d_{Q,1} + d_{1,2}) < Exp_1 - Exp_2 = \mathrm{diff}_{1,2} \tag{6}$$

where

$$\mathrm{diff}_{1,2} = \begin{cases} [100,200] - [200,400] = [-300, 0] & \text{if 1 is src,} \\ [200,400] - [100,200] = [0, 300] & \text{if 2 is src,} \\ [200,400] - [200,400] = [-200, 200] & \text{otherwise.} \end{cases}$$

Similarly, we derive the following from inequalities for V_{t_3}:

$$\begin{aligned} d_{Q,3} - (d_{Q,1} + d_{1,3}) &< \mathrm{diff}_{1,3} \\ d_{Q,3} - (d_{Q,2} + d_{2,3}) &< \mathrm{diff}_{2,3}. \end{aligned}$$

If we assume system 1 to be the source, and for a conservative solution we choose the minimum value of diff, we get

$$\begin{aligned} \min(\mathrm{diff}_{1,2}) &= \min(\mathrm{diff}_{1,3}) = -300 \\ \min(\mathrm{diff}_{2,3}) &= -200. \end{aligned}$$

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 $d_{Q,1} = d_{Q,2} = d_{Q,3} = 100$ ms, we get: $d_{1,2} > 300$, $d_{1,3} > 300$, and $d_{2,3} > 200$. These delays exhibit worst case behavior for the *timer suppression mechanism*.

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 repond to a request from system Q in the interval $[C_1 * E_{i,Q}, (C_1 + C_2) * E_{i,Q}]$, 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.)

Ref. [4] suggests values or C_1 and C_2 as 1 or $log_{10}G$, where G is the number of members in the group. We take $C_1 = C_2 = 1$ to synthesize the worst case topology. We get the expression

$$Exp_1 - Exp_2 = \left[\frac{(d_{1,Q} + d_{Q,1})}{2}, d_{1,Q} + d_{Q,1}\right] - \left[\frac{(d_{2,Q} + d_{Q,2})}{2}, d_{2,Q} + d_{Q,2}\right]. \quad \text{(A.2)}$$

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

Substituting in (A.2), we get $d_{1,2} > 100$ ms. Under similar assumptions, we can obtain $d_{2,3} > 100$ ms, and $d_{1,3} > 100$ ms.

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 V 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 (such as LINDO) can be used to obtain solutions for larger topologies.

B. Timer Configuration

In this section, 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 $[x, y]$ ms. We show an example for worst case analysis.

1) Worst Case Analysis: If the given ranges for the delays are $[2, 200]$ ms for all delays, then the term $d_{Q,j} - d_{Q,i} + d_{j,i}$ evaluates to $[-196, 398]$. From (A.2), we get $Exp_i < Exp_j -$ 196, to guarantee that a response is triggered.

If the delays are $[5, 50]$ ms, we get $Exp_i < Exp_j - 45$, i.e., i's expiration timer must be less than j's by at least 45 ms. 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.

ACKNOWLEDGMENT

The authors would like to thank Y. Yu and A. Cerpa for their help in LINDO and some *ns-2* simulations.

REFERENCES

[1] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast-sparse mode (PIM-SM): Protocol specification," IETF, RFC 2362, June 1998.

[2] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei, "Protocol independent multicast-dense mode (PIM-DM): Protocol specification," IETF, Proposed RFC, Sept. 1996.

[3] B. Cain, S. Deering, I. Kouvelas, W. Fenner, and A. Thyagarajan, "Internet group management protocol, version 3," IETF IDMR, RFC 3376, Oct. 2002.

[4] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Trans. Networking*, vol. 5, pp. 784–803, Dec. 1997.

[5] K. Miller, K. Robertson, A. Tweedly, and M. White, "StarBurst Multicast File Transfer Protocol (MFTP) specification," IETF, Internet Draft, 1998.

[6] R. Govindan, H. Yu, and D. Estrin, "Large-scale weakly consistent replication using Multicast," Univ. Southern California, USC-CS-TR-98-682, Sept. 1998.

[7] H. Yu, L. Breslau, and S. Shenker, "A scalable Web cache consistency architecture," in *Proc. ACM SIGCOMM*, 1999, pp. 163–174.

[8] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web caching: Toward a new global caching architecture," *Comput. Networks ISDN Syst.*, vol. 30, no. 22–23, pp. 2169–2177, Nov. 1998.

[9] M. Handley, "Session directories and scalable Internet multicast address allocation," in *Proc. ACM SIGCOMM*, Sept. 1998, pp. 105–116.

[10] E. Amir, S. McCanne, and R. Katz, "An active service framework and its application to real-time multimedia transcoding," in *Proc. ACM SIGCOMM*, Sept. 1998, pp. 178–189.

[11] A. Reddy, R. Govindan, and D. Estrin, "Fault isolation in multicast trees," in *Proc. ACM SIGCOMM*, Jan. 2000, pp. 29–40.

[12] J. Atwood, O. Catrina, J. Fenton, and W. Strayer, "Reliable multicasting in the Xpress transport protocol," in *Proc. 21st Local Computer Networks Conf.*, Oct. 1996, pp. 202–211.

[13] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Network Working Group, RFC 1889, Jan. 1996.

[14] A. Helmy and D. Estrin, "Simulation-based STRESS testing case study: A multicast routing protocol," in *6th Int. Symp. Modeling, Analysis and Simulation of Computer And Telecommunication Systems (MASCOTS '98)*, July 1998, pp. 36–43.

[15] A. Helmy, D. Estrin, and S. Gupta, "Fault-oriented test generation for multicast routing protocol design," in *Proc. IFIP TC6/WG6.1 Joint Int. Conf. (FORTE/PSTV'98)*, Paris, France, Nov. 1998, pp. 93–109.

[16] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, H. Yu, and Y. Xu, "Advances in network simulation," *IEEE Computer*, vol. 33, pp. 59–67, May 2000.

[17] K. Calvert, M. Doar, and E. Zegura, "Modeling Internet topology," *IEEE Commun. Mag.*, vol. 35, pp. 160–163, June 1997.

[18] K. Varadhan, D. Estrin, and S. Floyd, "Impact of network dynamics on end-to-end protocols: Case studies in reliable multicast," in *Proc. 3rd IEEE Symp. Computers and Communications* , 1998, pp. 147–153.

[19] P. Huang, D. Estrin, and J. Heidemann, "Enabling large-scale simulations: selective abstraction approach to the study of multicast protocols," in *6th Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)*, July 1998, pp. 241–248.

[20] A. Helmy, D. Estrin, and S. Gupta, "Systematic testing of multicast routing protocol robustness: Analysis of forward and backward search techniques," in *IEEE Int. Conf. Computer Communications and Networks (IC3N)*, Oct. 2000, pp. 590–597.

[21] S. Begum, M. Sharma, A. Helmy, and S. Gupta, "Systematic testing of protocol robustness: Case studies on mobile IP and MARS," in *Proc. IEEE Conf. Local Computer Networks (LCN)*, Nov. 2000, pp. 369–380.

[22] K. Seada, S. Gupta, and A. Helmy, "Systematic evaluation of multicast congestion control protocols," in *Proc. SCS Int. Symp. Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2002, pp. 857–867.

[23] K. Seada and A. Helmy, "Fairness evaluation experiments for multicast congestion control protocols," in *Proc. IEEE GLOBECOM*, Nov. 2002, pp. 2614–2618.

[24] K. Saleh, I. Ahmed, K. Al-Saqabi, and A. Agarwal, "A recovery approach to the design of stabilizing communication protocols," *J. Comput. Commun.*, vol. 18, pp. 276–287, Apr. 1995.

[25] E. Clarke and J. Wing, "Formal methods: State of the art and future directions," in *Proc. ACM Workshop Strategic Directions in Computing Research*, vol. 28, Dec. 1996, pp. 626–643.

[26] R. Boyer and J. Moore, *A Computational Logic Handbook*. Boston, MA: Academic, 1988.

[27] J. Spivey, *Understanding Z: A Specification Language and its Formal Semantics*. Cambridge, U.K.: Cambridge Univ. Press, 1988.

[28] F. Lin, P. Chu, and M. Liu, "Protocol verification using reachability analysis," *Comput. Commun. Rev.*, vol. 17, no. 5, pp. 126–135, Oct. 1987.

[29] P. Godefroid, "Using partial orders to improve automatic verification methods," in *Proc. 2nd Workshop Computer-Aided Verification*, 1990, pp. 176–185.

[30] F. Pong and M. Dubois, "Verification techniques for cache coherence protocols," *ACM Computing Surveys*, vol. 29, no. 1, pp. 82–126, Mar. 1996.

[31] M. Yannakakis and D. Lee, "Testing finite state machines: Fault detection," *J. Comput. Syst. Sci.*, vol. 50, no. 2, pp. 209–227, 1995.

[32] D. Rayner, "OSI conformance testing," *Comput. Networks ISDN Syst.*, vol. 14, no. 1, pp. 79–98, 1987.

[33] K. Sabnani and A. Dahbura, "A new technique for generating protocol tests," *ACM Comput. Commun. Rev.*, vol. 15, no. 4, pp. 36–43, Sept. 1985.

[34] ——, "A protocol test generation procedure," *Comput. Networks ISDN Syst.*, vol. 15, p. 285297, 1988.

[35] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. New York: IEEE Press, 1994.

[36] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, "Network visualization with Nam, the VINT network animator," *IEEE Computer*, vol. 33, pp. 63–68, Nov. 2000.

[37] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Scalable coordination in sensor networks," *Proc. Mobicom*, pp. 263–270, Aug. 1999.

[38] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, pp. 373–395, 1984.

[39] G. Dantzig, *Simplex Method for Solving Linear Programs*. London, U.K.: Macmillian, 1987.

[40] B. Borchers and J. Mitchell, "An improved branch and bound algorithm for mixed integer nonlinear programs," *Comput. Oper. Res.*, vol. 21, no. 4, pp. 359–367, 1994.

[41] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Motivation and architecture," IETF, Proposed RFC, Oct. 1996.

Ahmed Helmy (S'94–M'99) received the B.S. degree in electronics and communications engineering and the M.S. degree in engineering math from Cairo University, Cairo, Egypt, in 1992 and 1994, respectively, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the University of Southern California (USC), Los Angeles, in 1995 and 1999, respectively.

Since 1999, he has been an Assistant Professor of electrical engineering at USC. In 2000, he founded and is currently directing the wireless networking laboratory at USC. His current research interests lie in the areas of protocol design and analysis for mobile ad hoc and sensor networks, mobility modeling, design and testing of multicast protocols, IP micromobility, and network simulation.

Dr. Helmy received the USC Zumberge Award in 2000, and the National Science Foundation (NSF) CAREER Award and the Best Paper Award from the IEEE/IFIP International Conference on Management of Multimedia Networks and Services (MMNS) in 2002. He has been a member of the Association for Computing Machinery since 1994.

Sandeep Gupta received the Bachelor's degree in electrical engineering from the Indian Institute of Technology, Kharagpur, India, in 1985, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Massachusetts, Amherst, in 1989 and 1991, respectively.

He is currently an Associate Professor in the Department of Electrical Engineering–Systems, University of Southern California, Los Angeles. His research interests are in the area of VLSI testing and design. He is currently involved in projects on test and validation of deep-submicron circuits, testing multicore systems-on-silicon, and delay testing and diagnosis of high-speed circuits. He is also involved in a project on testing and verification of network protocols.

Dr. Gupta is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. He was a recipient of the National Science Foundation's Research Initiation Award in 1992 and CAREER Award in 1995. He was also a recipient of the Northrop Grumman Assistant Professorship in 1995 and the Zumberge Fellowship in 1996 at the University of Southern California. He received the Honorable Mention Award from the International Test Conference in 1997 and the Best Paper Award from the Asian Test Symposium in 2000.

Deborah Estrin received the Ph.D. degree in computer science from the Massachusetts Institute of Technology (MIT), Cambridge, in 1985.

She is currently a Professor of computer science at the University of California at Los Angeles (UCLA) and Director of the Center for Embedded Networked Sensing (CENS), a newly awarded National Science Foundation Science and Technology Center. She was on the faculty of the Department of Computer Science at the University of Southern California (USC), Los Angeles, from 1986 through mid-2000. During the subsequent ten years, her research focused on the design of network and routing protocols for very large, global, networks. She has been instrumental in defining the national research agenda for wireless sensor networks, first chairing a 1998 DARPA ISAT study and then a 2001 NRC study; the latter culminated in the NRC publication *Embedded Everywhere: A Research Agenda for Networked System of Embedded Computers*. Her research group develops algorithms and systems to support rapidly deployable and robustly operating networks of thousands of physically embedded devices. She is particularly interested in applications to environmental monitoring.

Dr. Estrin received the National Science Foundation Presidential Young Investigator Award for her research in network interconnection and security in 1987 at USC. She has served on numerous program committees and editorial boards, including the ACM SIGCOMM, Mobicom, SOSP, and the IEEE/ACM TRANSACTIONS ON NETWORKING. She is a Fellow of the Association for Computing Machinery and of the American Association for the Advancement of Science.