A Framework for Systematic Evaluation of Multicast Congestion Control Protocols

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

***Abstract*—Congestion control is a major requirement for multicast to be deployed in the current Internet. Due to the complexity and conflicting tradeoffs, the design and testing of multicast congestion control protocols is difficult. In this paper, we present a novel framework for systematic testing of multicast congestion control protocols. In our framework, we first design an appropriate model for the studied protocols based on the protocols specifications and correctness conditions, and then we develop an automated search engine to generate all possible error scenarios and filter these errors to come up with a selected set of scenarios that we evaluate in more detailed simulations. Our methodology helps in identifying the potential problems of the studied protocols and points to possible improvements. We hope that this will provide a valuable tool to expedite the development and standardization of such protocols.**

***Index Terms*—Congestion control, multicast, protocol modeling, simulation, systematic protocol testing, transmission control protocol (TCP) friendliness.**

I. Introduction

THE GROWTH of the Internet and its increased heterogeneity has increased the complexity of network protocol design and validation. The need for a systematic method to study and evaluate network protocols is becoming more important. Such methods aim to expedite protocol development and improve protocol robustness and performance. The systematic testing of robustness by evaluation of synthesized scenarios (STRESS) methodology [12]–[14] provides a framework for systematic design and testing of network protocols. STRESS uses a finite-state machine model of the protocol and search techniques to generate protocol scenarios that exercise the protocols in manners that expose errors or worst-case performance.

In this paper, we present a novel framework for testing multicast congestion control protocols based on the STRESS methodology. The goal of this framework is the generation of scenarios to evaluate protocol correctness, performance, and fairness. The evaluation targets specific protocol mechanisms that are incorporated in current and proposed protocols. Our aim is to have an abstract evaluation, applicable to a class of protocols. For illustration, we have chosen, as a case study, a multicast congestion control scheme called *pgmcc* [21], which is a single-rate scheme designed to be fair to transmission control protocol (TCP). However, we evaluate the protocol behavior based on protocol mechanisms and building blocks, so that our results can be generalized to other protocols that use similar mechanisms (such as TFMCC [26]).

The motivation and contribution of this work lies mainly in two areas. First, extending the STRESS methodology by applying it to a complex problem that has multiple dimensions: multicast, reliability, and congestion control (with the related issue of TCP friendliness). STRESS has been applied earlier to multicast routing and transport protocols, but congestion control is a new application that necessitates consideration of new semantics to the methodology, such as modeling of sequence numbers, traffic, and congestion window. This problem is also characterized by a high degree of interleaving between events and long-term effects of individual events. New challenges are faced due to the large size of the search space, and are handled by developing new types of equivalences and modifications to the search strategy to reduce its complexity. A mechanism for error filtering is developed to cope with the large number of error scenarios generated.

The second motivation and contribution is to provide a tool to aid in the design of multicast congestion control protocols. Multicast transport has been an area of extensive research and many protocols have been proposed [8], [17]. One of the most important criteria that Internet Engineering Task Force (IETF) [16] requires a multicast protocol to satisfy is to perform congestion control in order to safely deploy the protocol in the Internet. It is believed that the lack of good, deployable, well-tested multicast congestion control mechanisms is one of the factors inhibiting the usage of Internet protocol (IP) multicast [26]. By providing a systematic framework for evaluating these protocols, we aim to expedite the development and standardization in this area. This tool is also targeted to application designers, so that if it is impossible to eliminate all problems potentially caused by a protocol, then there should be at least a systematic way to enumerate them, and study their effects on the applications that use the protocol, so that the application can be designed to work around these problems.

Using our methodology, we are able to generate scenarios that reveal the problems in the studied protocols. Some of our interesting results include the effect on throughput due to joining and leaving of receivers, the effect of feedback suppression on congestion control, and the effect of changes in the special receivers representing the group. We validate these generated scenarios using detailed $ns2$ [3] simulations. In addition, based on our systematic study, we provide a set of experiments that can be used as a benchmark to evaluate the fairness of multicast congestion control mechanisms when running with competing TCP flows. We select the experiments to target specific congestion control mechanisms and to reveal the differences between

Manuscript received August 23, 2003. This work was supported in part by grants from the Defense Advanced Research Projects Agency (DARPA).

The authors are with the Electrical Engineering Department, University of Southern California, Los Angeles, CA 90089 USA (e-mail: seada@usc.edu; helmy@usc.edu; sandeep@usc.edu).

Digital Object Identifier 10.1109/JSAC.2004.836013

TCP and the proposed multicasting protocol. Several congestion control mechanisms are targeted by the experiments such as timeouts, response to acknowledgment (ACKs) and losses, and the effect of independent and congestion losses. In addition, we evaluate multicast mechanisms such as the effect of multiple receivers, group representative selection, and feedback suppression. Our analysis shows some strengths and potential problems of the studied mechanisms and point to possible improvements. Some scenarios reveal TCP unfriendly behavior, due to high losses or feedback suppression. Also, poor performance, due to group representative switch has been observed.

The rest of this paper is organized as follows. In Section II, we provide a background about multicast congestion control and a brief description of pgmcc. Section III gives a brief background of STRESS and an overview of our methodology. In Section IV, we present the protocol model that we use and the errors that we examine. In Section V, we show our search technique and its complexity. Section VI shows the problems identified. Detailed simulations for some of the identified problems in addition to the fairness experiments are provided in Section VII. Conclusions and future work are presented in Section VIII.

II. Multicast Congestion Control

The design of a multicast congestion control protocol that provides high performance, scalability, and TCP friendliness is a difficult task that attracts a lot of research effort. Multicast congestion control can be classified into two main categories: single-rate and multirate. Single-rate has a limited scalability because all receivers must receive data at the same rate (that of the slowest receiver). It also suffers from feedback implosion and drop-to-zero [2] (where the rate degrades significantly due to independent losses by a large number of receivers) problems. Multirate, where different receivers can receive at different rates, is more scalable but it has other concerns such as the complex encoding of data, possible multiple paths in the layered approach, and the effects of receivers joining and leaving layers.

TCP-friendly multicast congestion control can also be classified into window-based and rate-based. Window-based protocols have similar congestion window control as TCP and they follow the same steps of TCP window increments and cuts in order to be fair, while rate-based protocols depend on the TCP throughput equation [19] for adjusting the transmission rate in a smoother way and they compute the average transmission rate over longer periods in order to be fair over the long term average. Rate-based protocols also require accurate round-trip time (RTT) computations, which is a hard task in multicast. For more details about these issues, see [10] and [25].

Another possible classification for single-rate protocols is whether or not they are representative-based. Nonrepresentative-based protocols solve the scalability problems using some aggregation hierarchy. This requires complex building of the hierarchy and may need network support. The performance is still limited and [4] shows that even without losses, small variations in delay can cause fast performance degradation with the increase in number of receivers. Representative-based protocols provide a promising emerging approach to solve the scalability problems, where a small dynamic set of receivers is responsible for providing the feedback [6]. The main challenge is the dynamic selection of representatives in a scalable and efficient manner with appropriate handling of changes in representatives. Examples of single-rate representative-based protocols are *pgmcc* (window-based) [21] and *TFMCC* (rate-based) [26]. As an illustrative case study, we shall use the pgmcc protocol. Note that other single-rate representative-based protocols (including TFMCC), share a lot of mechanisms with pgmcc and, hence, our results also applies to the mechanisms in those classes of protocols. We have chosen pgmcc, since it is well-publicized and has been thoroughly studied using traditional methods. In the rest of this section, we will provide a more detailed description of pgmcc.

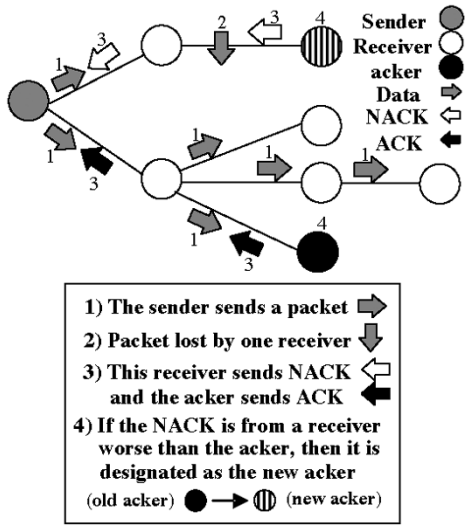

Fig. 1. Sample pgmcc scenario.

pgmcc [21] is a single-rate multicast congestion control scheme that is designed to be TCP-friendly. It provides both scalability and fast response. To achieve fast response while retaining scalability, a group representative called the *acker* is selected and a tight control loop is run between it and the sender. It is called the acker because it is the receiver that sends the ACKs. Other receivers can send nNegative acknowledgments (NACKs) when they lose packets, if a reliable transport protocol is used.[1] pgmcc has been used to implement congestion control in the PGM protocol [23]. A sample scenario for pgmcc is shown in Fig. 1.

The acker is the representative of the group. It is chosen as the receiver with the worst throughput to ensure that the protocol will be TCP-friendly. A window-based TCP-like controller based on positive ACKs is run between the sender and the acker. The feedback in pgmcc is provided in receiver reports that are used by the sender to estimate the throughput. They are embedded into the NACKs and ACKs and contain the loss rate and information for computing an estimate for the RTT of the sending receiver.

[1]pgmcc can be used with both reliable and nonreliable transport protocols. Nonreliable protocols will also need to send NACKs from time to time for congestion control purposes.

The most critical operation of pgmcc is the acker selection and tracking. As mentioned, the receiver with the worst throughput is selected as the acker. When another receiver with worse throughput sends a NACK, an acker change may occur. A receiver that does not send NACKs is assumed to have no congestion problems, and will not be considered. Computation of throughputs uses information sent by receivers and the TCP-like formula: $T \propto 1/\mathrm{RTT}\sqrt{p}$, where T is the throughput, RTT is the round-trip time estimate, and p is the loss rate [19]. An acker switch occurs from receiver j to receiver i if $T(i) < c\,T(j)$, where $T(n)$ is the throughput computed by the sender for receiver n. c is a constant $(0 < c \leq 1)$ that is used during throughput comparison to dampen oscillations of acker selection when the difference between the current acker and the worst receiver is not large. There is a 32-bit field in the ACK called the bitmask, which indicates the receive status of the most recent 32 packets. This is included to help the sender deal with lost and out-of-order ACKs.

A window based congestion control scheme similar to that used by TCP is run between the sender and the acker. The parameters used are a window W and a token count T. W is the number of packets that are allowed to be in flight and has the same role as the TCP window, while T is used to regulate the sending of data packets by decrementing T for every packet sent and incrementing it for every ACK received. On packet loss, W is cut by half and T is adjusted accordingly. A packet is assumed lost when it has not been acked in a number (normally three) of subsequent ACKs. W and T are initialized to 1 when the session starts or after a stall when ACKs stop arriving and a timeout occurs. pgmcc has a different flow/reliability window than that used for congestion to decouple congestion control from retransmissions, so it can be used with both reliable and nonreliable protocols.

Some multicast transport protocols depend on router support for feedback aggregation. For example, in PGM the first instance of a NACK for a given data segment is forwarded to the source, and subsequent NACKs are suppressed. During the study of the protocol, there are different variations to consider. The main two variations are 1) reliable versus nonreliable transport and 2) with feedback aggregation (router support) versus without feedback aggregation.

A related work is shown in [18], where the Maude formal methodology [7] is applied to specify and analyze the AER/NCA [15] reliable multicast protocol. The benefit of using Maude is that it is a rewriting logic language that supports a variety of modeling and formal methods. In this paper, we are using a different procedural approach based on the STRESS methodology. In our approach, we tailor the model and search based on the specific study to improve the efficiency of covering the search space and generating the error scenarios.

III. Methodology Framework

STRESS provides a framework for systematic design and testing of network protocols. It uses a finite-state machine (FSM) model of the protocol and search techniques to generate protocol tests. The generated tests include a set of network scenarios each of which leads to violation of protocol correctness and behavioral requirements. In [11] and [12], STRESS has been used to verify multicast routing protocols using forward and backward search techniques. In [13] and [14], it is used for the performance evaluation of timer suppression mechanisms in multipoint protocols. In [1], mobile IP and multicast over asynchronous transfer mode (ATM) are studied. Currently, we also use STRESS for the verification of wireless medium access control (MAC) protocols. In this paper, we extend STRESS to study multicast congestion control. Congestion control is a new application that necessitates consideration of new semantics, such as modeling of sequence numbers, traffic, and congestion window. Multicast congestion control is characterized by a high degree of interleaving between events and long-term effects of individual events. New challenges are faced due to the large size of the search space, and are handled by developing new types of equivalences and modifications to the search strategy to reduce complexity. A mechanism for error filtering is developed to cope with the large number of error scenarios generated. In the remainder of this section, we give a high-level description for our methodology and how different blocks described throughout the paper fit within.

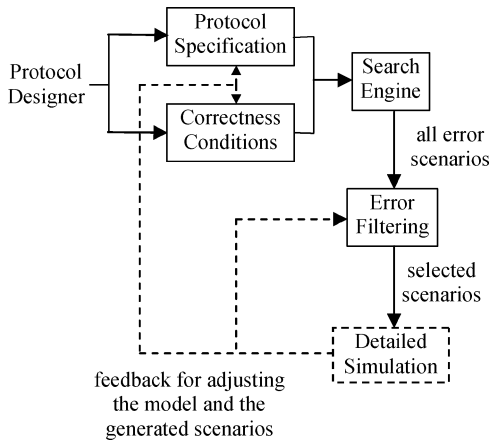

Fig. 2. Block diagram of our methodology.

Fig. 2 shows the main framework. The protocol designer or tester inputs the protocol specifications and correctness conditions in the form of a state model (Section IV-A), a transition table (Section IV-B), and an error model (Section IV-C). The search engine (Section V) explores the search space for errors and generates a set of error scenarios. These error scenarios are fed into the error filter which classifies the scenarios and outputs a compact set of error scenarios that can be used for testing the protocol. These scenarios are then further validated through more detailed, packet-level, simulations to ensure that the difference between the somewhat abstract STRESS model and the much more detailed simulation code does not hide the undesirable performance shown by the scenarios. Simulations also help in assessing and quantifying the effects of these scenarios on performance. The feedback from the simulations can be used by the designer to adjust the input and the filtering. Currently the search engine and error filtering are automated.

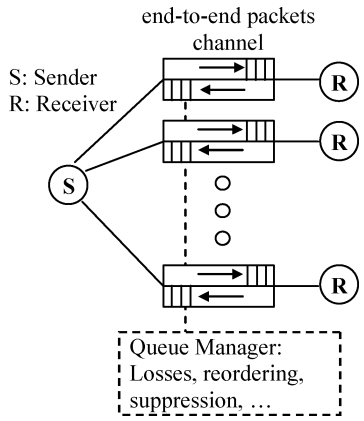

Fig. 3. Topology model: the path between the sender and a receiver is represented by a virtual end-to-end channel. The topological details (delays, losses, suppression, and routing effects) are captured by taking all permutations of loss and packet reception events.

IV. Protocol Model

In order to process the protocol and apply the STRESS methodology, we need to have a suitable model that represents the protocol and the system. The model is built according to the protocol specification and additional assumptions based on our understanding of the protocol behavior and its context. The model consists of the following:

- a representation of the protocol states;
- a state transition table defining the possible events and their effects on states;
- a set of errors that represents our correctness criteria.

Critical to our methodology is how the topology is modeled. A multicast transport session topology consists of a sender, a group of receivers, links, and routers. This detailed topology can be very complex for our model and search. Also, many of the details are unimportant. For a multicast congestion control protocol operation, the topological details are reflected mainly on which packets are received by the sender and receivers, and when these packets are received. Our goal here is to simplify the topology and still capture characteristics as delays, reordering, selective and correlated losses, and feedback suppression.

Fig. 3 shows the simplified topology. A virtual end-to-end channel exists between the sender and each receiver. We are able to capture all the required characteristics with this topology by using all permutations of events (this will become clearer when we explain the search) including losses, reception of packets in different orders, and suppression. The order in which packets are received or dropped in this topology emulate the delays, routing effects, and feedback aggregation in detailed topologies. This simplification may hide some details, but we believe such details do not affect our protocol evaluation. To further verify the validity and effect of our generated scenarios, we perform packet-level $ns2$ simulations with more detailed topology models (see Section VII).

A. States

The global state is an aggregation of local states of a sender and a number of receivers, with a separate end-to-end channel between the sender and each receiver, as shown in the topology in Fig. 3. Each entity has a number of state parameters that define its local state.

- Sender state: packets sent, ACKs received, sequence number of next packet to send, sequence number of next ACK expected, current acker, current acker throughput, window size, number of tokens, duplicate count (to count the duplicate ACKs), ignore count (for adjusting the token count after a window cut).
- Receiver state: receiver ID, packets received, sequence number of next packet expected, loss ratio, multicast group member or not.
- A Packet channel between sender and each receiver, such that each packet in the channel has the following parameters:
  - Packet: type (data, ACK, or NACK), sequence number, acker (if it is a data packet), packets received by acker (if it is an ACK), loss ratio (if it is an ACK or NACK), missing packet (if it is a NACK)

The channel holds the packets between the sender and the receiver with data going in one direction and feedback (ACK or NACK) going in the other. The underlying network characteristics, such as packet losses, packet reordering (e.g., due to route changes), and network delays, are handled by the channel. Multicast group membership is represented by defining at a state whether each receiver is a group member or not. It is assumed that an underlying multicast routing protocol exists and delivers the packets to current members only. In addition to the combination of these local states, we have also added a *global loss estimator* to our global state, in order to estimate the loss rates on different channels independent of local computations by receivers. As will be shown later, this is required only for protocol evaluation.

B. Transition Table

A simplified version of this protocol's transition table is shown in Table I. The transition table describes, for each possible event, the conditions for its occurrence, and the effects of that event. For example, *Send* is the event of sending a packet, its condition is that the number of tokens is above 1, and its effects are putting the packet in the group member receivers' channels and adjusting the sender parameters. During the search process the conditions of these events are checked at every state, and if satisfied the corresponding event is triggered and a new state is generated. Slow start is ignored in our model, since it is a transient phase, and we are more interested in testing the steady state behavior. Some changes can be made to the transition table to study variations. For example, the retransmission of packets can be omitted to consider nonreliable transport. NACK suppression is emulated by NACK dropping.

C. Errors Model

Table II shows the errors we check for in our model. These are potential problems that should be considered during the implementation of the protocol or by the applications using it. Some of these errors violate the correctness such as "gaps in sequence space," which means that, for some reason, the sender

TABLE I
SIMPLIFIED TRANSITION TABLE FOR THE PROTOCOL

Event	Precondition	Effect (Transition)
Send	Tokens >= 1	-Decrement number of tokens (T=T-1) -Add packet to sent list and increment the next sequence to send -Add packet to all members' channels
Join	Receiver not member	-Receiver becomes member
Leave	Receiver member	-Receiver becomes non-member
Loss	Packet in channel	-Remove packet from channel -If it is a data packet, update global loss estimation for that channel.
Receive Data	Data packet in channel	-Remove data packet from channel and update global loss estimation -If that receiver is a member, update its receive list, next sequence to receive, loss ratio -If lost packets discovered, generate and send NACKs, update loss ratio -If that receiver is acker send ACK
Receive ACK	ACK in channel	-Remove ACK from channel -If ACK from current acker, update its throughput using its loss ratio and RTT -Update ACK list and sequence of next ACK expected -Increment window and token (W=W+1/W, T=T+1+1/W) -If packet losses discovered (duplicate ACKs), cut window (W=W/2) and adjust token (ignore next W/2 ACKs)
Receive NACK	NACK in channel	-Remove NACK from channel -Compare that receiver throughput with acker throughput, and if NACK is from a receiver worse than acker then choose it as new acker -Retransmit

TABLE II
ERRORS CHECKED IN OUR MODEL

Error	When to check	How to check
Unnecessary starvation	At the end of search or at timeout	Tokens<1 and no more data or ACKs in channels
Wrong acker selection	At sending (or at the end of search)	Comparing acker and receivers throughput (by global loss estimators)
Wrong congestion notification	At congestion (when packet loss detected)	Check whether packets are really lost (using global loss estimator)
Duplicate packets	At data reception by receiver	Check receive list for if the packet already exist
Gaps in sequence space	At the end of search	Check sent list of the sender for gaps

has not sent all the packets in sequence. Most of the other errors mainly cause performance degradation such as "unnecessary starvation," which means that the sender has packets to send but it is stalled, although the group receivers are ready. Duplicate packets means that a receiver received the same packet more than once, which is a waste of the network bandwidth. Wrong congestion notification can lead to an unnecessary window cut and reduction in sending rate. An error that leads to fairness violation is "wrong acker selection," which means that a receiver

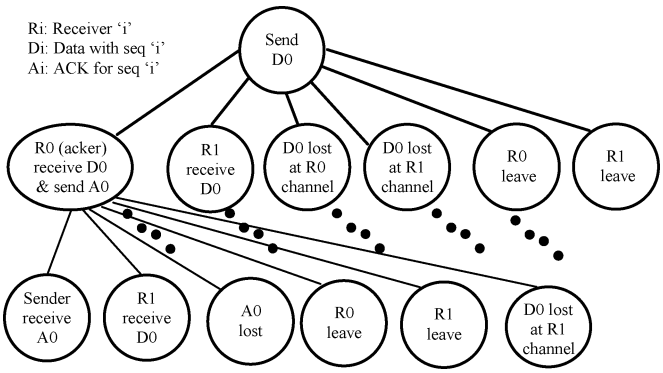

Fig. 4. Example of search space expansion.

other than the worst receiver is chosen as acker. Since in this protocol we consider fairness (TCP friendliness) as being synonymous to choosing the worst receiver as acker, "wrong acker selection" models the fairness violation.

During the search, we look for erroneous states by checking for these errors conditions. The point at which an error is checked is important, since this has a significant impact on the search time and on the number of erroneous states generated. It also affects the number of false errors. We want to reduce the error checking as much as possible by considering only the minimum set of states required to discover the error. For example, whenever possible, we check for errors only at the end of the search path (leaves of search tree). This is possible for errors that become permanent once they appear. Whether an error is permanent or not depends on the events that may change its conditions. For example, "unnecessary starvation" is a permanent error and can be checked at the end, since it leads to a stall, but if we include timeouts in our model, then timeout will be the event that can eliminate the starvation condition, and we will need to check for that error at timeouts.

V. SEARCH PROBLEM

The problem of test synthesis can be viewed as a search problem. By searching the possible sequence of events over possible network topologies and checking for errors at specific points, we construct the test scenarios that stress the protocol. Forward search is used to investigate the protocol state space. As in reachability analysis, forward search starts from initial states and applies the stimuli (events) 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. Of course, in our case, we cannot expand forever, so we limit our search to a maximum sequence number, which is the number of packets to be sent by the sender. The search process is completely automated.

At each state during the search, the preconditions for events are checked, and transitions are applied to generate new states. A list of all visited (already expanded) and working (still to be expanded) states is kept, so that new states can be checked against it to avoid redundant search. Error checking is performed at specific states during the search according to the type of error. An example of a part of the search tree is shown in Fig. 4. Notice

TABLE III
NUMBER OF VISITED STATES WITH TWO RECEIVERS. THE LAST COLUMN SHOWS THE NUMBER OF VISITED STATES WHEN EQUIVALENCES (SEE SECTION V-C) ARE EXPLOITED

Seq	Complete	Without reordering	Without losses	Without join & leave	Without all	Complete with equivalences
1	64	64	52	16	7	16
2	384	264	260	96	16	96
3	28264	3820	11372	7066	40	6082
:						
20					14520	

TABLE IV
NUMBER OF COMPARISONS WITHOUT EQUIVALENCES AND HASHING

seq \ rcvr	1	2	3
1	109	4650	2.00e
2	311	1.48e	1.50e
3	17891	1.16e	

that this leads to state space explosion and techniques need to be used to reduce the complexity of the space to be searched.

A. Search Complexity

To get a measure of the complexity we are facing and to understand the source of this complexity, we show in Table III the number of visited states in a topology of two receivers and increasing sequence number (the limit of the search). By using all the events (complete interleaving), we find that the number of visited states increases at a very high rate with the increase in sequence numbers. The next three columns show the reduction in the number of visited states by performing the search without packet reordering, packet losses, and members join and leave, respectively. (Packet reordering means that the channel delivers packets in all possible orders, so that at each state there is an event triggered for each packet to be received independent of its location in channel. Removing reordering means that packets in the same channel and same direction arrive in order, but interleaving still exist between packets in different channels or different directions.) In the next column, we show that without all these three types of events, we can reach 20 sequence numbers with a number of visited states about half of those reached with three sequence numbers and complete interleaving. These three events are considered as undesirable events in our search. (Another reason for considering them undesirable is that they are the main cause of errors.) Tables IV and VI show the total number of comparisons and the number of transitions and their large increase with the increase in receivers and sequence numbers.

Using complete interleaving and applying the equivalences explained in Section V-C, the search can cover up to four sequence numbers. By analyzing the protocol mechanisms, we found that four sequence numbers are enough to detect the protocol errors in our model. The reasoning behind that is included in the technical report [22].

B. Comparison Time

In order to avoid redundancy in state generation, a list of all visited (already expanded) and working (still to be expanded) states is kept, against which new states can be checked. The comparison time is the most time consuming process during the search, since the comparison time for a state is proportional to the number of visited and working states at that time. As the number of states increases, the comparison time becomes extremely large, which limits the number of sequence numbers that can be covered. Some statistics are shown in Table IV. To reduce the comparison time and expand our search space, we use hashing. A hash table is used for storing all the states (visited and working); the working list is still used for the search process only.

C. Equivalences

Exhaustive search has exponential complexity and the number of states generated is very large. To reduce this complexity, we use the notion of equivalence. The notion of equivalence implies that by investigating an 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. We use three types of equivalence: state equivalence, path equivalence, and error scenarios equivalence. Error scenarios equivalence is provided by the error filtering approach used to reduce the number of generated erroneous scenarios. State equivalence means that two states are not exactly equal, but they are equivalent from our viewpoint. Path equivalence is to find paths that lead to the same states and prune them, this can cause a major reduction in complexity as will be seen. Table V shows the number of visited states when equivalences and hashing are used withd with the direct search results in Table IV. Table VII shows the reduction in number of transitions due to equivalences compared with Table VI.

1) State Equivalence: Examples of state equivalences used in our case study are *channel equivalence* and *receiver equivalence*. Channel equivalence is when two global states have similar local states and similar packets in their channels but in different order. They are considered equivalent, since we consider all orderings of packet arrival. Receiver equivalence occurs when several receivers have identical local states. In this case, e.g., sufficient to apply the transitions for only one of them, since the same search subtrees will be generated for others. For

TABLE V
NUMBER OF COMPARISONS WITH EQUIVALENCES AND HASHING

seq \ rcvr	1	2	3
1	2	16	108
2	4	71	2887
3	57	18632	

TABLE VI
NUMBER OF TRANSITIONS WITHOUT EQUIVALENCES

seq \ rcvr	1	2	3
1	22	220	1880
2	42	1568	6.7e4
3	423	1.8e5	

TABLE VII
NUMBER OF TRANSITIONS WITH EQUIVALENCES

seq \ rcvr	1	2	3
1	7	25	81
2	14	147	1931
3	101	10514	

example, if we have two receivers R1 and R2 with the same parameters, we do not need to apply the same events for both of them, since this will just create identical states with R1 and R2 swapped. One way to implement that is to use descriptors to represent channels and receiver states where the packet order and receiver ID are ignored in the descriptor representation. A more general approach is to use a global descriptor for representing the whole state taking all the state equivalences into account.

2) Path Equivalence: Path equivalences are based on the notion that if two events do not affect each other, then it is not necessary to consider interleaving between them. Detecting path equivalences can be very complex, but it leads to large reductions in search space. One way to identify path equivalences is to look at the different events in the transition table and see how they affect each other, so that we need not consider interleaving between events that do not affect each other directly. An example is shown in Fig. 5, which is the search tree for one receiver and one sequence number. Notice that there are a lot of redundant paths. Consider for example the leave and join events. After a leave, you need to consider the join only if there is a change (send or receive) in the channel. Also, after a join, you need to consider the leave only if there is a change in the channel. A loss of a packet is affected only by the receiving of that packet, which means that repeating it on all paths is redundant.

Taking these equivalences into account, we see in Fig. 6 that the complexity is reduced, and with higher number of receivers

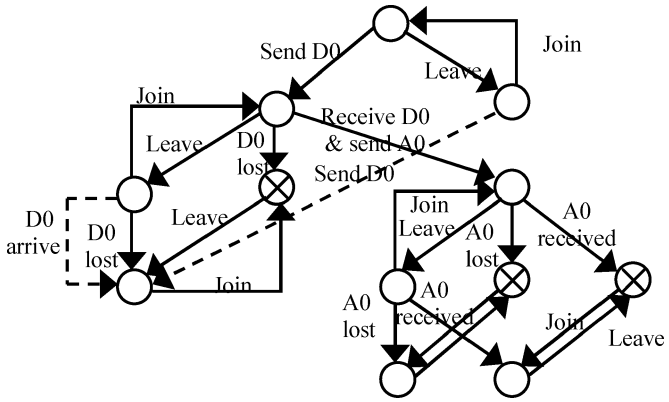

Fig. 5. Search tree for one receiver and one sequence number without path equivalence (number of visited states: 12, number of transitions: 22, and number of comparisons: 109).

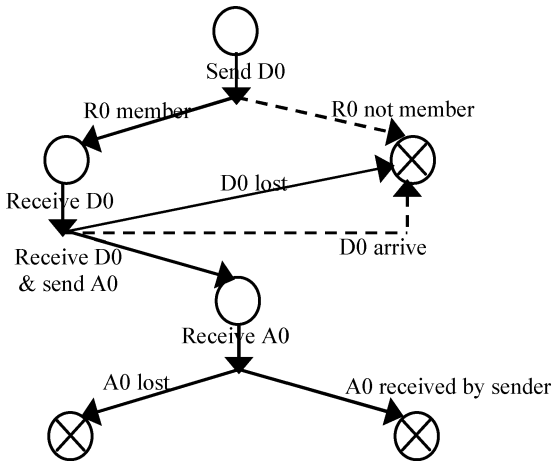

Fig. 6. Search tree for one receiver and one sequence number with path equivalence (number of visited states: 6, number of transitions: 7, and number of comparisons: 19).

and sequence numbers the reduction is greater. In Fig. 6, the join and leave events of a receiver are associated with sending and receiving of packets on the channel of that receiver and the loss event is triggered only once under the same send subtree. So, we have two major events (send and receive) and other events are internally associated with them. Larger reduction can be obtained by restricting the interleaving between packets received by different receivers. A receiver is affected only by changes occurring on its channel, and not by other receivers' channels (although the effect of that on the sender is taken into account during the sending of new packets). So instead of taking the permutations of packet arrival of all packets in all channels at a certain state, we need to take only the permutations of each individual channel with the combinations between receivers. This reduces the number of different packet arrival orders from $\sum_{k=0}^{l}\binom{l}{K}k!$, where $l=\sum_{i=1}^{m}n_i$ is the total number of packets in m channels (m is the number of receivers, and n_i is the number of packets in channel i), to $\prod_{i=1}^{m}\sum_{k=0}^{n_i}\binom{n_i}{k}k!$, e.g., in the case of two receivers ($m=2$), with $n_1=3$ and $n_2=2$ packets in their channels, the number of different packet arrivals reduce from 326 to 80.

TABLE VIII
SUMMARY OF DETECTED PROBLEMS

- Unnecessary starvation
  - All data to Acker lost
  - All ACKs lost
  - Some data, some ACKs lost
  - Acker leaves (crashes)
- Duplicate packets
  - False NACKs due to out-of-order (OOO) data causing retransmissions
  - Sender or receiver crashes (non-deterministic)
- Wrong congestion notification (wrong window cut)
  - OOO data packets
  - Acker switch causing OOO ACKs
- Wrong Acker selection
  - Very bad Acker (all data or ACKs lost), this can lead to wrong Acker selection after timeout
  - Very bad receiver (data lost, ACKs or NACKs lost)
  - NACK suppression
  - False NACKs
  - Wrong loss ratio estimate by receiver due to leaving and joining
- Gaps in sequence space
  - Sender or receiver crashes (non-deterministic)

D. Error Filtering and Hiding

Two contradicting challenges faced during the generation of the error scenarios[2] are the large number of generated error scenarios representing only a restricted set of errors and the hiding of some errors by other errors. We provide two algorithms for filtering the error scenarios into classes of distinct errors. Due to space constraints the details are included in the technical report [22].

VI. DETECTED PROBLEMS

Using our methodology, we are able to generate scenarios that lead to several problems. In Table VIII, we summarize the problems detected. Many of these problems have not been noted before, such as the unnecessary starvation problems, the effect of representative switching and the effect of receivers joining and leaving causing the selection of wrong representatives. Even, for the known problems, they are here detected in a systematic way using an automated tool. In the following subsections, we explain these problems for the studied protocol model. In the technical report [22], we give a hint on generalizing the results to the building blocks of various protocols.

A. Unnecessary Starvation

We define unnecessary starvation as a scenario where the sender has packets but it is not able to send, although the group receivers are ready to receive more packets. In practice, the state may remain stalled until the sender starts sending again, e.g., after a timeout. This leads to a reduction in throughput and longer delays experienced by receivers. There are two types of scenarios that can lead to unnecessary starvation. The first type is when all the data to acker or the ACKs from acker are lost, since the sender depends on the ACKs for generating new packets. This is an expected problem and, according to the protocol definition, in this case the rate should follow the worst receiver. The other scenario that can lead to unnecessary starvation is when the group representative (the acker in our case study) leaves the group without notification (or crashes) and the sender gets no feedback. The sender will wait for some time (until a timeout) before start sending again and switching to another representative.

[2]An *error scenario* is a sequence of states (and events) that ends with an *error state* based on our error model.

B. Wrong Acker Selection

Wrong acker selection means that a receiver other than the worst receiver is selected as the acker. This leads to fairness violation and affects the TCP friendliness of the protocol. Several scenarios can lead to wrong acker selection. If there is a very bad receiver such that it is not even able to send feedback, another receiver will be selected as the acker. The other receiver may have a higher rate, which makes things worse on the bad receiver path. This is complementary to the unnecessary starvation case. In unnecessary starvation, the problem is on the multicast session itself, while here we are concerned about competing traffic. Simulations in Section VII show some interesting tradeoffs between the two cases.

Out-of-order packets causing false NACKs can also lead to wrong acker selection. For example, if data packets arrive at a receiver out-of-order (e.g., due to route change), the receiver sends NACKs for the missing packets with a high loss ratio. This loss ratio may cause the sender to designate it as the acker, although it is a good receiver. The effect of this scenario will depend on how loss ratio is computed in the case of out-of-order packets, how long the receiver waits before sending the NACK, and what happens to the loss ratio after the receiver discovers that the packets were not lost. In many cases, such specific details are not specified or even articulated during protocol design.

Some multicast transport protocols suppress NACKs to avoid the NACK implosion problem. NACK suppression can cause the NACKs of worse receivers to be suppressed, especially if they are far away, which leads to wrong acker selection. In the simulation section, we show how this problem affects TCP friendliness.

Another type of scenarios that can lead to wrong acker selection is where the loss ratio is estimated incorrectly by receivers leaving and joining the group. If a receiver leaves the group and then joins again after a while, the loss ratio estimate by that receiver will not be an accurate estimate of the current channel situation. Moreover, it may consider packets not received as lost in computing the loss rate, although they were not lost. This problem can have a large effect on the operation of the protocol, if receivers are not stable or if there are problems in the underlying multicast routing and group membership protocol. Since the throughput computation depends on the local loss ratio computed by receivers, inaccuracy in that value can significantly affect the protocol behavior. In pgmcc, the throughput is used only for acker selection, but in rate-based protocols that depend on throughput computations for determining the sending rate such as TFMCC [26], this problem can have a larger impact.

C. Wrong Congestion Notification

Wrong congestion notification means that the sender receives a congestion notification (duplicate ACKs) and cuts its window, although there is no congestion and this reduction in sending

rate is not necessary. This can occur due to out-of-order data arrival at the receiver causing wrong interpretation of loss. Another scenario that leads to wrong congestion notification is when an acker switch occurs between receivers with large difference in delay. This leads to ACKs arriving out-of-order, which can cause severe performance degradation if not handled properly by the sender as will be shown by the simulations.

D. Duplicate Packets

In these scenarios, a receiver receives the same packet more than once, which wastes the network bandwidth. A scenario leading to duplicate packets is when data packets arrive out-of-order at the receiver causing it to send NACKs for packets still on their way. Adding some delays in sending the NACKs by receiver and the retransmissions by sender can reduce this effect, but there will be a tradeoff between the delay value and the responsiveness in case of packet losses. Another reason for duplicate packets is sender or receiver crashes ending in a nondeterministic state.

E. Gaps in Sequence Space

Gaps in sequence space means that the sender is not sending packets in sequence, which can also occur due to crashes leading to nondeterministic states. Sender crashes may be considered unlikely, and if they occur the session can be restarted. However receiver crashes have a high probability as the number of receivers increase. Hence, we need to make sure that malicious behavior by a single receiver will not affect the entire session. For example, if a receiver sends an ACK or NACK with a sequence number greater than the highest sequence number sent by the sender, how the sender will respond to that? Such malicious behavior can be due to a nondeterministic state caused by a receiver crash or may be an intentional denial of service attack. We believe that such cases should be considered if the protocol is to be scaled to large number of receivers.

VII. Simulations

To show the utility of our methodology, we have conducted a set of detailed simulations for the generated error scenarios. These simulations validate the generated scenarios and demonstrate how the stepwise errors detected lead to noticeable performance problems and long-range effects. We provide also a set of experiments targeted specifically to evaluate the fairness of a multicast congestion control protocol when running with competing TCP flows. The experiments presented are constructed directly from the generated scenarios except the experiments of Sections VII-C and VII-D, where the scenarios are based on details of TCP flavors and the TCP equation that are not in our model.

We use $ns2$ [3] for performing the experiments. The source models used in the simulation are FTP sources with packet size of 1400 bytes. The links have propagation delay of 1 ms, and bandwidth of 10 Mb/s, unless otherwise specified. The queues have a drop-tail discard policy (RED is also examined and the results are similar) and first-in–first-out (FIFO) service policy, with capacity to hold 30 packets. For TCP sessions, both Reno and SACK are examined and they support similar conclusions

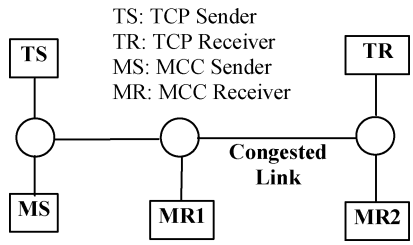

Fig. 7. A MCC session with two receivers MR1 and MR2, where MR2 is the worst receiver (accordingly the acker) due to the congested link. A TCP session is also competing for the congested link.

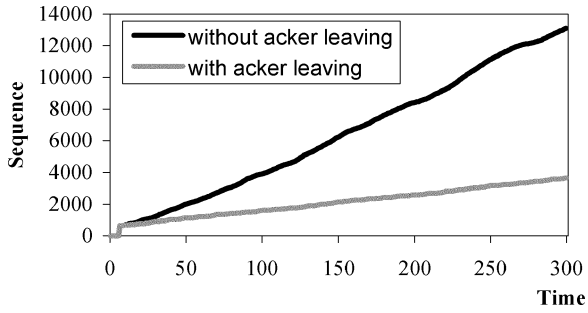

Fig. 8. Throughput of pgmcc without and with acker leaving.

(except for the timeouts experiment, as will be shown). In the graphs, we show the sequence numbers sent by the sender versus the time. This has the same effect as showing the throughput. We provide a fairness metric f as the ratio between TCP and pgmcc throughput.[3] For more details about the experiments, see also [22].

A. Unnecessary Starvation

A generated scenario that leads to unnecessary starvation is that of the acker leaving the group (or crashing), as was shown in Section VI-A. Guided by that scenario, we build the topology in Fig. 7, where we have a pgmcc session with a sender and two receivers. The path to the receiver MR2 has a congested link (1 Mb/s), hence, MR2 is the worst receiver and becomes the acker. A TCP session is added to generate extra traffic over that path. The experiment runs for 300 s. Every 10 s, the acker MR2 leaves the group for 1 s. During this period, the sender receives no ACKs and is not able to send any packets until it timeouts. This leads to drastic reduction in throughput by about 75% as shown in Fig. 8.

B. Wrong Acker Selection

Incorrect loss ratio estimation by receivers, due to leaving and joining the group, can lead to wrong acker selection as detected by our methodology (Section VI-B). We again use the topology in Fig. 7, but this time it is the other receiver MR1 that leaves the group for 1 s, every 10 s. After rejoining MR1 can have an incorrect estimation of the loss ratio, due to the packets missed when it had left for which it starts sending NACKs. The sender may select MR1 as the acker instead of MR2. Fig. 9 shows the throughput of pgmcc and TCP without the receiver leaving and

[3]The final sequence numbers in the graphs represent the aggregate throughput. So their ratio can be considered as the ratio between the average instantaneous throughputs.

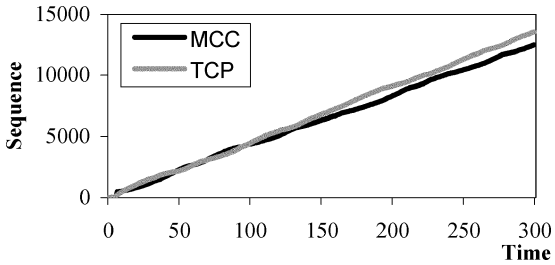

Fig. 9. Throughput of pgmcc and TCP without acker switch ($f = 108\%$).

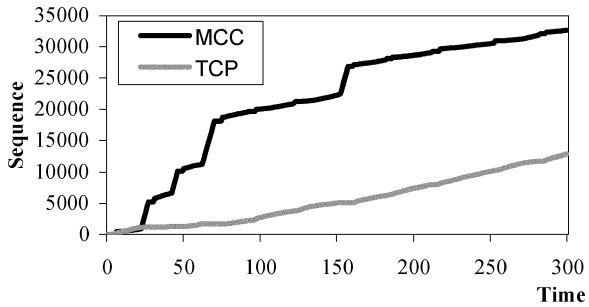

Fig. 10. Throughput of pgmcc and TCP with wrong acker switch ($f = 39\%$).

joining (no acker switching). Fig. 10 shows the effect of receiver leaving and joining on the throughput. In that figure, the rate of pgmcc increases highly during the periods of wrong acker selection (when MR1 becomes the acker), which is seen in the high slope of the throughput. The low slope represents the periods where the correct acker (MR2) is chosen, which is close to the TCP slope. Although the sender throughput is high, these packets are received by MR1, but MR2 throughput is much less. At the high rate periods the congested link is not able to transfer all packets to MR2 and, hence, it loses these packets and in a reliable session they have to be retransmitted.

C. Window and Timeouts

This set of experiments contains simple topologies to compare the MCC protocol with different flavors of TCP. The flavors are Reno, New-Reno, and SACK [9]. This comparison helps us understand the behavior of the protocol and the subtle differences between it and TCP. Two congestion control issues are targeted by this comparison: 1) reaction to losses and ACKs with its effect on the window size and 2) retransmission timeouts. TCP Reno is still the most widely deployed flavor in the Internet, but recent statistics show that TCP New-Reno and TCP SACK deployment is increasing [20]. New-Reno and SACK solve performance problems of TCP in case of multiple-packet loss in a window and they reduce the number of timeouts. When multiple packets are lost from a single window of data, New-Reno and SACK can recover without a retransmission timeout. With Reno and New-Reno at most one dropped packet is retransmitted per RTT, while SACK does not have this limitation [9]. This response to losses and ACKs has a major impact on the window size, and consequently on the fairness. According to [19], timeouts also have a significant impact on the performance of TCP Reno and they constitute a considerable fraction of the total number of loss indications. Measurements have shown that in

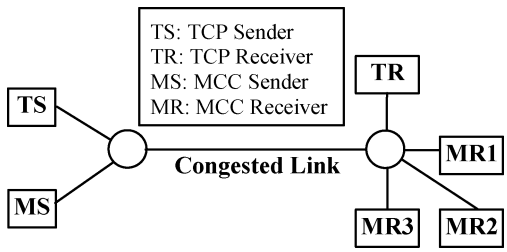

Fig. 11. TCP session competing with MCC session over a bottleneck link.

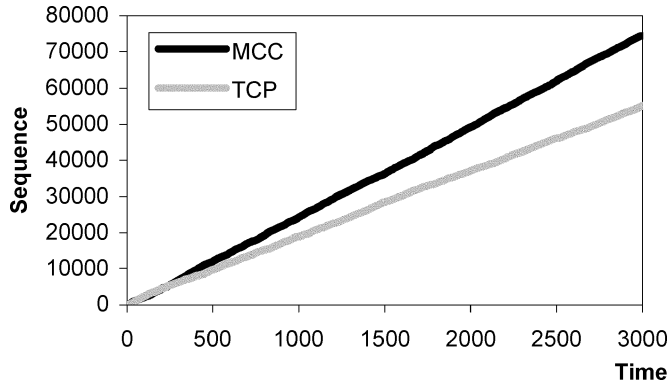

Fig. 12. Throughput of pgmcc versus TCP Reno ($f = 74\%$).

many cases the majority of window decreases are due to timeouts, rather than fast retransmits. This experiment highlights the protocol policy in handling ACKs and timeouts, and which flavor it is closer to.

We use the topology shown in Fig. 11 to test the fairness of pgmcc with the different TCP flavors in a very simple case, where we have only a single TCP session competing with a pgmcc session over a bottleneck link (500 kb/s, 50 ms). pgmcc has a number of identical receivers, so any one of them could be the acker. Starting with TCP Reno and comparing the throughput of the TCP sender with the pgmcc sender, we find in Fig. 12 that pgmcc is not fair to TCP Reno.[4] pgmcc times out after a stall when the ACKs stop coming in, and a long fixed timeout expires. Without timeout pgmcc reacts to congestion by cutting the window in half similar to fast recovery in TCP. TCP, on the other hand, adjusts its timeout value depending on the measured RTT and the variance of the measured RTT values. In addition, ACKs in pgmcc are per-packet as in SACK, while in Reno ACKs are aggregate only, so for Reno to send an ACK for a packet, all packets in between have to be received. This has a large effect when multiple packets are dropped from a window.

Our explanation of the unfairness that is observed over long periods is due to these differences in handling timeouts and responding to ACKs and losses. By observing the window size changes in both of them (Fig. 13), we found that the pgmcc window is larger most of the time and it does not enter the slow start phase. We have also conducted several other experiments for different timeout values, and found that the results obtained depend heavily on this value. The appropriate value for timeout that achieves fairness depends on dynamic network conditions

[4]This experiment runs for 3000 s. When we run the experiment for 300 s, as in [17], the unfairness shown here was not clear.

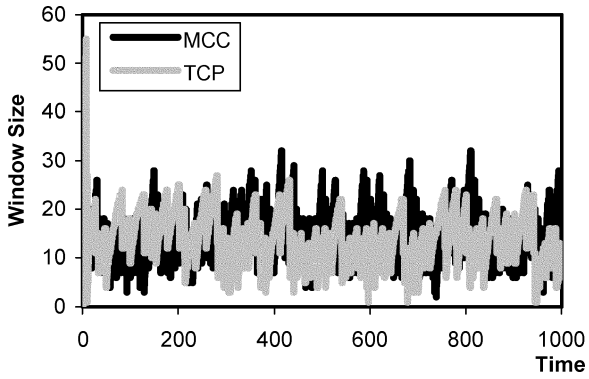

Fig. 13. Window size comparison of pgmcc and TCP Reno.

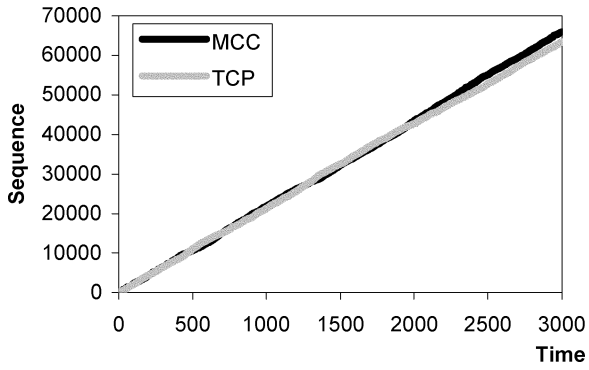

Fig. 14. Throughput of pgmcc with the adaptive timeout versus TCP Reno ($f = 96\%$).

that change over time. We then run the same experiments with New-Reno and SACK. New-Reno and SACK reduce the timeouts and solve the performance problems when multiple packet are dropped from a window. Simulation results show that pgmcc is fairer ($f = 92\%$) with SACK and New-Reno.

To further clarify the effect of timeout and window size, we run the same experiment of TCP Reno with an adaptive timeout mechanism added to pgmcc similar to that used in TCP and the reset of the timeout is controlled to be as close as possible to TCP Reno. It is reset only if there are no packets missing in the received bit mask (i.e., all packets are acked). Because of differences in RTT between different ackers, after a switch a fixed timeout is used until the adaptive timeout for the new acker is computed. Fig. 14 shows the result of pgmcc compared with TCP Reno after adding the adaptive timeout. The modified pgmcc is friendly to Reno.

D. Diverse Losses and Delay

This set of experiments addresses the effect of having multiple receivers with different losses and delay. The throughput of the MCC protocol when the receivers have different combinations of delay and loss rates (e.g., high loss, low delay versus low loss, high delay) is compared with the competing TCP flows. There are several objectives behind this comparison: First, better understanding of the effect of losses, retransmissions, and delays with multiple receivers. Second, many MCC protocols use a TCP throughput equation to model the TCP behavior. This set of experiments evaluates the accuracy of the used equation.

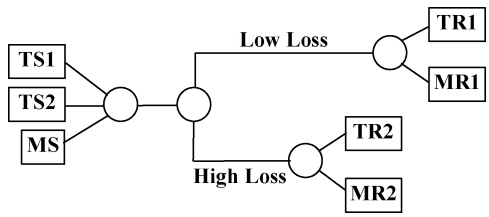

Fig. 15. MCC session with receivers having different delays and loss rates competing with TCP sessions.

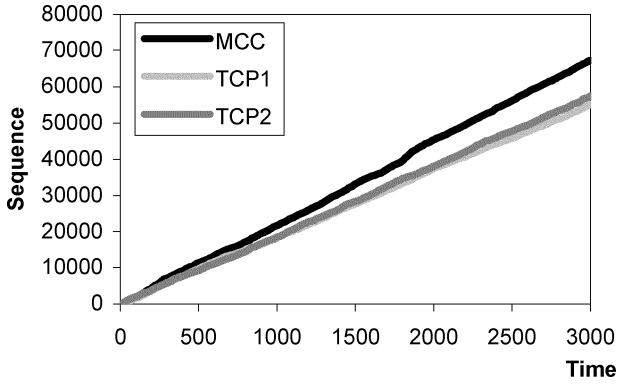

Fig. 16. Throughput of pgmcc versus the two TCP sessions with low loss rate ($f1 = 82\%$, $f2 = 85\%$).

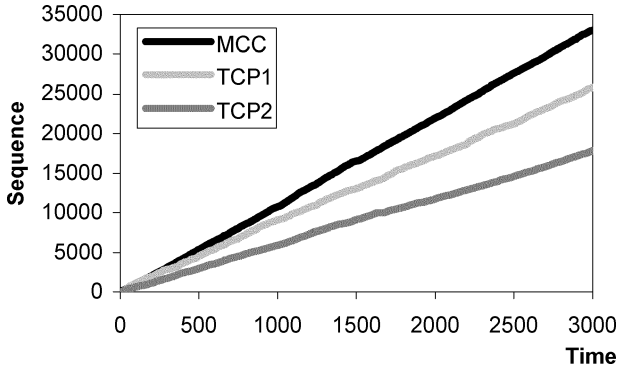

Fig. 17. Throughput of pgmcc versus the two TCP sessions with high loss rate ($f1 = 78\%$, $f2 = 54\%$).

In Fig. 15, we have two pgmcc receivers, one with high RTT (400 ms) and low loss rate (0.4% or 2%) and the other with lower RTT (200 ms) and higher loss rate (1.6% or 8%). In Fig. 16, we see that pgmcc and the two TCP sessions have close throughput.[5] The loss rates here are 0.4% for the low loss link and 1.6% for the high loss link. In Fig. 17, we are using a loss rate of 2% for the low loss link and 8% for the high loss link, which causes pgmcc to be unfair to the high loss rate TCP session. This is mainly because of the difference in how these protocols react to packet losses. In pgmcc, the reliability window is separated from the congestion window and the handling of acknowledgment is different. Unlike TCP, in pgmcc, the sender keeps sending new packets, even if previously lost packets have not been received. In addition, according to [19], the simplified equation used for computing the throughput is

[5]We set the parameters of RTT and loss rates to let the two TCP sessions get the same throughput, according to the TCP equation.

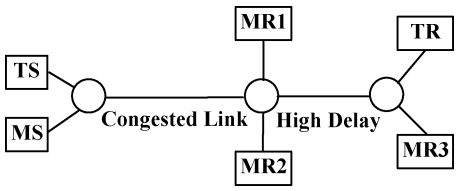

Fig. 18. MCC session with receivers having the same loss rate, but different delays.

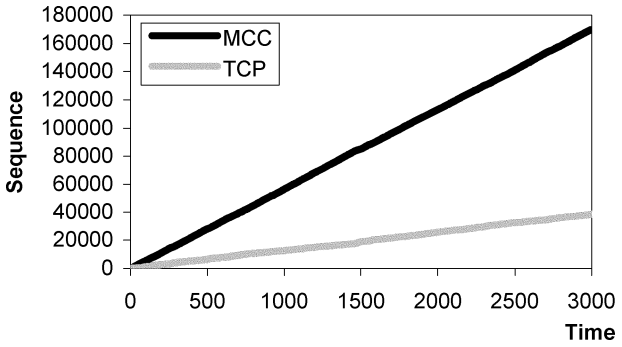

Fig. 19. Throughput of pgmcc versus TCP with NACK suppression ($f = 23\%$).

for fast retransmission only and it does not take timeouts into account. It also overestimates the throughput for losses above 5% and so it is suitable only when loss rates are below 5% and no timeouts happen. This experiment shows that at high loss rates pgmcc can be unfair to TCP due to the ignoring of previously lost packets by congestion control, and due to the inaccuracy in the throughput equation.

E. Feedback Suppression

Most MCC protocols depend on the receivers' feedback in making decisions. Some multicast transport protocols have network support (e.g., by routers) to improve their performance. This support is normally in the form of feedback aggregation or suppression to avoid problems such as ACK and NACK implosion. We consider experiments that test the effect of feedback suppression on fairness. One of the generated scenarios in Section VI-B shows how feedback suppression leads to wrong acker selection. Based on that scenario, we present experiments consisting of topologies with critical receivers having their feedback suppressed. The definition of critical receivers depends on the protocol. Feedback suppression affects the accuracy of decisions. These experiments investigate the tradeoff between the amount of feedback and the correctness of decisions or computations. In PGM [23], with feedback aggregation, the first NACK for a given data segment is forwarded to the source and subsequent NACKs are suppressed. Using the topology in Fig. 18, feedback aggregation will cause pgmcc to be unfair to TCP as can be seen in Fig. 19, because the worse receiver MR3 will always have its NACKs suppressed (the link leading to MR3 router has 50-ms delay).

This experiment shows that feedback suppression can cause pgmcc to be unfair to TCP. Accordingly, we propose a low overhead solution for that problem by random suppressing of NACKs in the router. The router will suppress NACKs only

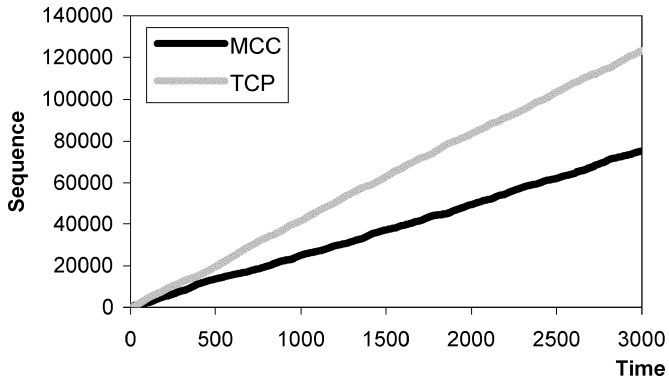

Fig. 20. Throughput of pgmcc versus TCP due to acker switches ($f = 164\%$).

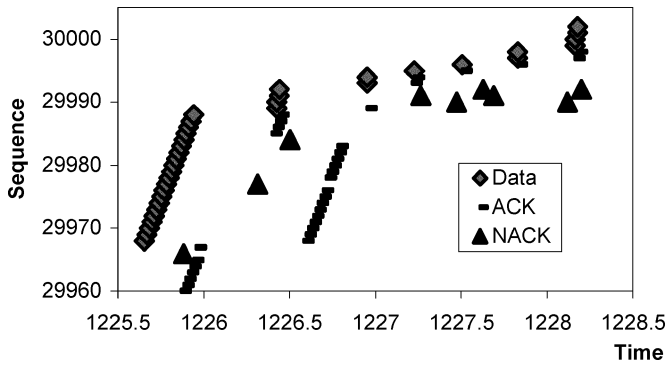

Fig. 21. Detailed sequence of pgmcc packets during acker change.

with some probability. This will give the worst receiver NACKs chances to reach the sender. There will be a tradeoff between the amount of feedback suppressed and the accuracy of acker selection and, hence, fairness with respect to TCP.

F. Group Representatives

Several MCC protocols use the idea of representatives to achieve scalability. Feedback is normally provided only by these special receivers. An important task is the selection and changing of the representatives. The experiments here target this operation by having configurations containing multiple sets of receivers that can be selected as representatives and having scenarios that trigger the changing between them. The aim of these experiments is to study the effect of these changes on the overall protocol operation and on its fairness to TCP.

In pgmcc, we evaluate the effect of acker switching using also the topology of Fig. 18, but with a higher delay (200 ms) in the link leading to the MR3 router. As shown in Fig. 20, the throughput of pgmcc becomes low, and the TCP throughput is higher. This does not constitute a fairness problem, but a performance degradation problem for pgmcc.

The reason for this poor performance is the acker switching between a high RTT receiver and a low RTT receiver similar to the error scenario generated in Section VI-C. By looking at acker switches in detail, we found that two switches happen in succession close to each other. The first NACK from the closer receiver causes an acker change, then the other NACK (upon its arrival) causes another change for the far one. This pattern repeats with packet losses. Fig. 21 shows more details and illustrates what happens between two acker switches (each vertical

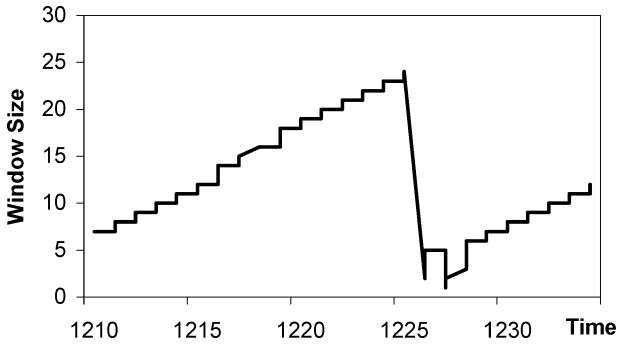

Fig. 22. Window size changes of pgmcc session during an acker change.

line indicates an acker switch). After the switch to the close receiver, new ACKs arrive before old ACKs. The old ACKs that arrive at 1226.5 do not cause new packets to be sent, i.e., they do not generate new tokens. Later, when new ACKs arrive the window starts at slow rate, which means that, it has been cut. Fig. 22 shows how the window is cut at 1226.5. This is due to the out-of-order ACK delivery and the consequent decisions taken by the sender. Wrong loss detections can be interpreted, because ACKs for old packets have not arrived yet. Also, on a loss detection, the sender tries to realign the window to the actual number of packets in flight, which will not be interpreted correctly after the switch, because there are still packets and ACKs in flight to and from the old acker.

To solve this problem the sender needs to keep track of the recent history of acker changes and the ACKs sent by each acker. In addition, the bitmask provides information about the recent packets received by the acker. Accordingly, the sender can adjust its window and avoid these problems.

This experiment shows that acker switching between receivers with large difference in delay degrades the performance of pgmcc. This problem will be more common on larger network topologies and group sizes.

VIII. Conclusion

We have described a framework for the systematic evaluation of multicast congestion control protocols based on the STRESS methodology. The application of congestion control extends and adds new semantics to STRESS, such as the modeling of sequence numbers, channels traffic, and congestion window. New challenges are faced due to the exponential increase in search space, the high level of interleaving between events, and the long-term effects of individual events. These challenges are handled by introducing new kinds of state and path equivalences. Also, several modifications are made to the search mechanism to provide more efficient coverage of the search space. Two algorithms are developed for error filtering to reduce the number of error scenarios generated.

Several problems have been discovered in our case study and a set of scenarios is generated. The generated scenarios can be used for testing multicast congestion control building blocks. Some of the interesting problems identified include the effect of receivers joining and leaving on the throughput measurements, the effect of NACK suppression on feedback, and the effect of having special receivers (such as the acker) which can leave or change. Identifying these problems in a systematic way is helpful for the protocol designer and for applications using that protocol.

We have validated our results through detailed packet-level simulations and presented a set of simulation experiments for fairness evaluation. Improvements are proposed to cope with some of the problems observed, such as random suppression of NACKs, sender response after representative switches, and the adaptive timeout in case fairness to TCP Reno is required.

The main limitation of our methodology is its high complexity, which constrains the search space covered and limits us to the detection of low-level errors. However, if not handled properly, these low-level errors can lead to high-level performance problems and long-range effects. Furthermore these errors are more likely to occur in larger topologies and real-life networks. As was shown by the simulations, these microscopic problems have large effect on performance, and can be used in generating higher level scenarios and guiding simulations of a larger scale in time and space.

References

[1] S. Begum, M. Sharma, A. Helmy, and S. Gupta, "Systematic testing of protocol robustness: Case studies on mobile IP and MARS," in *Proc. 25th Annual IEEE Conf. Local Computer Networks (LCN)*, FL, Nov. 2000, pp. 369–380.

[2] S. Bhattacharyya, D. Towsley, and J. Kurose, "The loss path multiplicity problem for multicast congestion control," in *Proc. IEEE INFOCOM'99*, New York, Mar. 1999, pp. 856–863.

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

[4] A. Chaintreau, F. Baccelli, and C. Diot, "Impact of network delay variation on multicast session performance with TCP-like congestion control," in *Proc. IEEE INFOCOM 2001*, Anchorage, AK, April 2001, pp. 1133–1142.

[5] D. M. Chiu, M. Kadansky, J. Provino, J. Wesley, and H. Zhu, "Pruning algorithms for multicast flow control," in *Proc. NGC2000*, Stanford, CA, Nov. 2000, pp. 83–92.

[6] D. DeLucia and K. Obraczka, "Multicast feedback suppression using representatives," in *Proc. IEEE INFOCOM'97*, Kobe, Japan, Apr. 1997, pp. 463–470.

[7] G. Denker, J. Meseguer, and C. L. Talcott, "Formal specification and analysis of active networks and communication protocols: The Maude experience," in *Proc. DARPA Information Survivability Conf. Exposition (DISCEX)*, 2000, pp. 251–265.

[8] C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint communications: A survey of protocols, functions, and mechanisms," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 277–290, Apr. 1997.

[9] K. Fall and S. Floyd, "Simulation-based comparison of Tahoe, Reno, and SACK TCP," *Comput. Commun. Rev.*, pp. 5–21, July 1996.

[10] S. J. Golestani and K. K. Sabnani, "Fundamental observations on multicast congestion control in the Internet," in *Proc. IEEE INFOCOM'99*, New York, Mar. 1999, pp. 990–1000.

[11] A. Helmy, D. Estrin, and S. Gupta, "Fault-oriented test generation for multicast routing protocol design," in *Proc. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV), IFIP*, Paris, France, Nov. 1998, pp. 93–109.

[12] ——, "Systematic testing of multicast routing protocols: Analysis of forward and backward search techniques," in *Proc. 9th Int. Conf. Computer Communications and Networks (IEEE ICCCN)*, Las Vegas, Nevada, October 2000, pp. 590–597.

[13] A. Helmy, S. Gupta, and D. Estrin, "The STRESS method for boundary-point performance analysis of end-to-end multicast timer-suppression mechanisms," *IEEE/ACM Trans. Networking*, pp. 44–58, Feb. 2004.

[14] A. Helmy, S. Gupta, D. Estrin, A. Cerpa, and Y. Yu, "Systematic performance evaluation of multipoint protocols," in *Proc. FORTE/PSTV, IFIP*. Pisa, Italy, Oct. 2000, pp. 189–204.

[15] S. Kasera, S. Bhattacharyya, M. Keaton, D. Kiwior, J. Kurose, D. Towsley, and S. Zabele, "Scalable fair reliable multicast using active services," *IEEE Network Mag. (Special Issue on Multicast)*, pp. 48–57, Jan./Feb. 2000.

[16] A. Mankin, A. Romanow, S. Bradner, and V. Paxson, "IETF criteria for evaluating reliable multicast transport and application protocols," RFC 2357, June 1998.

[17] K. Obraczka, "Multicast transport mechanisms: A survey and taxonomy," *IEEE Commun. Mag.*, pp. 94–102, Jan. 1998.

[18] P. C. Ölveczky, M. Keaton, J. Meseguer, C. Talcott, and S. Zabele, "Specification and analysis of the AER/NCA active network protocol suite in real-time Maude," in *Proc. Fundamental Approaches to Software Engineering*, 2001, pp. 333–348.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM 1998*, Vancouver, BC, Canada, Sept. 1998, pp. 303–314.

[20] J. Padhye and S. Floyd, "On inferring TCP behavior," in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 287–298.

[21] L. Rizzo, "pgmcc: A TCP-friendly single-rate multicast congestion control scheme," in *Proc. ACM SIGCOMM 2000*, Stockholm, Sweden, Aug. 2000, pp. 17–28.

[22] K. Seada, A. Helmy, and S. Gupta, "A framework for systematic evaluation of multicast congestion control protocols," Univ. Southern California, Los Angeles, CA, Tech. Rep., Sept. 2003.

[23] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano, "PGM reliable transport protocol specification," RFC 3208, Dec. 2001.

[24] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby, "Reliable multicast transport building blocks for one-to-many bulk-data transfer," RFC 3048, Jan. 2001.

[25] J. Widmer, R. Denda, and M. Mauve, "A survey on TCP-friendly congestion control," *IEEE Network Mag.*, pp. 28–37, May 2001.

[26] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," in *Proc. ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001, pp. 275–286.

Karim Seada (M'99) received the B.S. and M.S. degrees with honors in computer engineering from Cairo University, Cairo, Egypt, in 1998 and 2000, respectively. He is currently working towards the Ph.D. degree in electrical engineering at the University of Southern California, Los Angeles.

In summer 2002, he was an Intern with Intel's Network Architecture Laboratory, Hillsboro, OR, working on IP mobility. His current research interests lie in the area of computer networks and distributed systems with emphasis on the robustness of geographic protocols in wireless networks, resource discovery in ad hoc and sensor networks, design and testing of network protocols, and multicast congestion control.

Mr. Seada is a member of the Association for Computing Machinery (ACM).

Ahmed Helmy 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, 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 National Science Foundation (NSF) CAREER Award in 2002. In 2000, he received the USC Zumberge Research Award, and in 2002, he received the Best Paper Award from the IEEE/IFIP International Conference on Management of Multimedia Networks and Services (MMNS).

Sandeep Gupta (S'87–M'90) received the Bachelors of Technology degree in electrical engineering from the Indian Institute of Technology, Kharagpur, 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 an Associate Professor in the Department of Electrical Engineering-Systems, University of Southern California (USC), Los Angeles. His research interests are in the areas 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 digital circuits. He is also involved in a project on testing and verification of network protocols. His teaching interests include teaching and curriculum development in the areas of VLSI design, test, and CAD—at the graduate level, as well as undergraduate level.

Dr. Gupta is a recipient of the National Science Foundation's CAREER Award (June 1995–May 1998) and the Research Initiation Award (June 1992–May 1995). He is also a recipient of the Northrop Grumman Assistant Professorship (1995) and Zumberge Fellowship (1996) at USC. He also received the Honorable Mention Award from the International Test Conference in 1997, and is coauthor of the Best Paper Award at the 2000 Asian Test Symposium. He is a member of the IEEE Computer Society. He is also an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS.