MCA: An End-to-end Multicast Congestion Avoidance Scheme with Feedback Suppression

Jiang Li, Shivkumar Kalyanaraman

{*lij6@cs, shivkuma@ecse*}*.rpi.edu*
Rensselaer Polytechnic Institute, Troy, NY 12180

Abstract

In this paper, we propose MCA, an end-to-end multicast congestion *avoidance* scheme with feedback suppression. Congestion avoidance [7] is different from congestion control in the sense that our scheme detects and responds to network congestion *without necessarily inducing packet loss*. Our scheme is a single-rate scheme and operates end-to-end, i.e. the sending rate is controlled by the source based on feedback from the most congested receiver and *does not expect packet marking or other support* from intermediate nodes. We design it to be robust under both lossless and lossy situations. Congestion is detected at receivers using the concept of *"accumulation"* (the number of buffered bits of a flow inside the network) and simple thresholding techniques proposed in our recent unicast work [22]. For the purposes of choosing representative (the most congested receiver) by the source and suppressing feedback by receivers, each receiver maintains its *Good Throughput Rate At Congestion* (G-TRAC) (the product of receiving rate during congestion epochs and $1 - f$, where f is congestion occurrence frequency). In this way, receivers do not need to *continuously* (either densely or sparsely) exchange packets with the sender (e.g. to measure RTT). Therefore, MCA is scalable for large-size groups. We evaluate the design and demonstrate the performance of MCA using detailed ns-2 simulations.

Key words: Multicast, congestion avoidance, feedback suppression, G-TRAC

1 Introduction

Multicast is the preferred transport mechanism for simultaneously transferring bulk data to multiple receivers. Numerous applications such as content distribution, streaming, multi-player games, multimedia multi-user chat/telephony,

[1] Jiang Li is now with Howard University

distance education can greatly benefit from multicast. In this paper, we propose MCA, an end-to-end rate-based multicast congestion *avoidance* scheme with feedback suppression. Congestion avoidance [7] is different from congestion control in the sense that our scheme detects and responds to network congestion without necessarily inducing packet loss. Our scheme is a single-rate scheme, i.e., the sending rate is regulated by the most congested receiver. There was a previous single-rate multicast congestion control scheme by DeLucia et al [9] that provides some preliminary congestion avoidance functions, as it uses the congestion detection mechanism proposed in TCP Vegas [3]. However, in that scheme, incipient congestion is only measured at source side for the paths between the source and representative receivers, while other receivers still detect congestion on other paths by packet loss. Therefore, it is a mixture of congestion control and avoidance, whereas our scheme offers *pure* congestion avoidance. Other examples of single-rate multicast congestion control schemes include PGMCC [18], TFMCC [20], our earlier work GSC [14] and LE-SBCC [19] and references within. These schemes are "congestion control" schemes in the sense that receivers wait for packet loss which is signalled back to the source as congestion indications. If bottlenecks provide packet marking support (similar to TCP-ECN [17]), packet loss may be avoided in the above schemes, albeit special support (i.e. packet marking) is necessary. There exist another distinct class of congestion control schemes which are *multi-rate* (eg: RLC, FLID-DL, FGLM, STAIR [21,6,4,5]). Since multi-rate schemes are greatly different from single-rate ones, we are not going to discuss them here, although we could extend the single-rate work to the multi-rate area in the future.

In brief, MCA has the following features:

(1) It is an end-to-end scheme, i.e. it does not require special support from inside the network, such as packet marking.
(2) It uses a new concept of *accumulation* instead of packet loss to detect congestion, and thus can react to incipient congestion.
(3) It provides efficient *non-timer-based* feedback suppression.
(4) State and computation complexity at both source and receiver side are $O(1)$.

In more details, MCA uses a new concept of *"accumulation"* and simple thresholding techniques proposed in our recent unicast work [22] to achieve congestion avoidance on a *purely* end-to-end basis, i.e., it does not require packet marking support from interior bottlenecks. Accumulation is the number of buffered bits of a flow inside the network. Our congestion model uses accumulation to detect congestion end-to-end as real queues are *being* built up. It is a generalization of TCP Vegas's [3] congestion detection technique.

Like Vegas [13,12], MCA is inherently *incompatible* with the TCP, because

MCA reduces sending rate upon *incipient* congestion when there may not be packet loss yet, while TCP reduces sending rate upon serious congestion when packets are lost. Therefore, it is very likely that when MCA reduces sending rate, TCP still tries to consume more bandwidth. At last, the sending rate MCA will be kept at an abnormally low level. One way to ensure compatibility is to have a packet marking scheme at bottlenecks that indicates congestion as queues build up [11]. Or, if routers can support multiple-class traffic by leveraging the "type of service" field in IPv4 header [16] or the "traffic class" field in IPv6 header [10], our scheme can be used with TCP in parallel. Consequently, the target networks of MCA are those can be configured to allow separate queues in routers, such as campus networks and intranets.

In this scheme, we use a representative for the rate control purpose at the source [2] side. That is, at any time, the source keeps record of the slowest (i.e. the most congested) receiver (named *Congestion Representative* (CR)), and adjusts the sending rate according to that receiver's feedback. At the same time, receivers themselves also suppress their feedback if necessary so as not to overwhelm the source. Both mechanisms make use of a new metric *Good Throughput Rate At Congestion* (G-TRAC). G-TRAC is defined as $r(1-f)$, where r is the receiving rate *during congestion epochs*, f is congestion occurrence frequency (to be discussed later). The receiver with the lowest average G-TRAC is chosen as the slowest receiver, i.e. CR. Other receivers suppress their feedback if their average G-TRACs are higher than that of CR. By using G-TRAC, receivers no longer need to *continuously* (either densely or sparsely) exchange packets with the sender (e.g. to measure RTT).

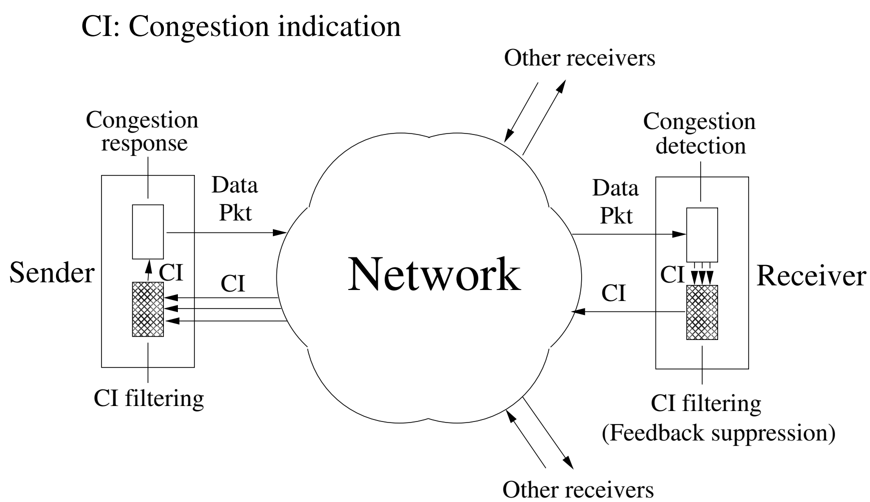


Fig. 1. MCA MODEL

MCA consists of four key building blocks (Figure 1), two at receivers and two at the source. The first block detects congestion at the receiver side using accumulation and possible (though rare) packet loss. The second block at the source

[2] In this paper, we use the terms *source* and *sender* interchangeably.

side responds to congestion. It implements an AIMD rate increase/decrease policy based on the CR's feedback. The third block, a filtering block at the receiver side, blocks *Congestion Indications* (CIs) generated by the congestion detection block if necessary. That is, it suppresses feedback. The fourth block, also a filtering block but at the source side, only allows CIs from CR through. All blocks only require small constant number of states and light computation.

Simulations show that MCA achieves high bottleneck utilization, while avoiding Drop-to-Zero problem [20,18,1]. Drop-to-Zero is the problem that reacting to *more feedback* than necessary leads to a beat-down of the multicast flow's rate[20,18,1]. This occurs because the multicast flow generates feedback on multiple paths and may not have them filtered sufficiently. Besides, there is the TCP-unfriendliness problem, which is the problem of reacting to *less feedback* than a hypothetical TCP flow would do on the worst loss path [2,18,20]. Since the congestion detection model is incompatible with that of TCP, we won't directly demonstrate the fairness with TCP. However, we demonstrate the fairness between multi-receiver and single-receiver MCA flows.

2 Concepts And Model

In MCA, congestion detection is based on the accumulation concept. In this section we define the accumulation concept using a bit-by-bit fluid model [8] [15], and develop an algorithm for measuring it in the multicast context at receiver side.

2.1 Accumulation

The concept of *accumulation* was first developed in our earlier unicast work [22]. We summarize the core ideas here. The discussion below assumes unicast fluid flows, but we extend it to multicast in a later section.

Consider an ordered sequence of FIFO nodes (routers) $\{R_1, \ldots, R_j, R_{j+1}, \ldots, R_J\}$ along the path of a *unidirectional* flow i in Figure 2(a). The flow comes into the ingress node R_1 and, after passing some intermediate nodes $R_2, \ldots, R_{J-1}$, goes out from the egress node R_J. At time t in any node R_j ($1 \leq j \leq J$), flow i's input rate is $\lambda_{ij}(t)$, output rate $\mu_{ij}(t)$. The propagation delay from node R_j to node R_{j+1} is d_j.

We define the arrival curve $A_{ij}(t)$ of a flow i at a node R_j as the number of bits from that flow which have cumulatively arrived at the node up to time t, and similarly the service curve $S_{ij}(t)$ as flow i's bits cumulatively serviced

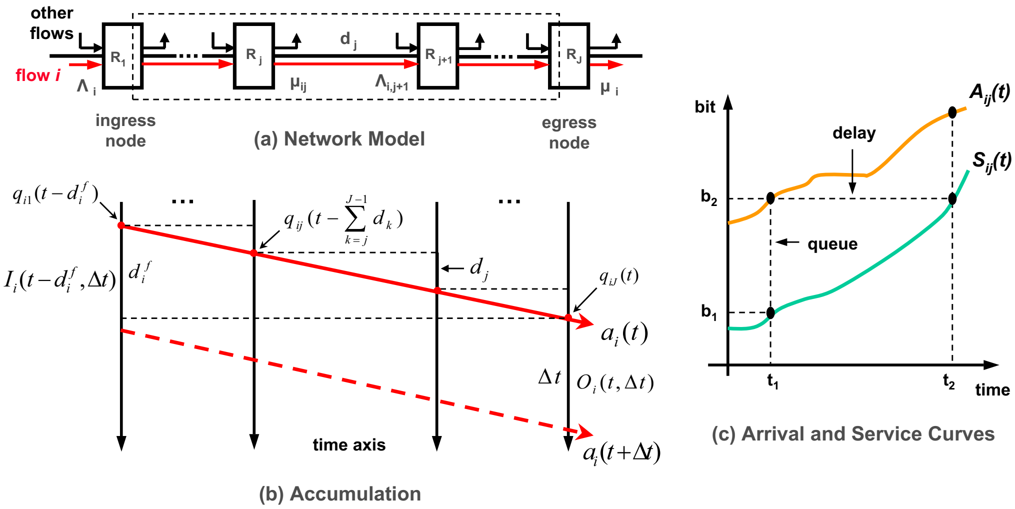


Fig. 2. NETWORK FLUID MODEL: ACCUMULATION CONCEPT

at node R_j [8] [15], drawn in Figure 2(c). For any FIFO node R_j, both $A_{ij}(t)$ and $S_{ij}(t)$ are continuous[3] and non-decreasing functions. If there is no packet loss, then at any time t, by definition, flow i's buffered bits $q_{ij}(t)$ in node R_j is the difference between $A_{ij}(t)$ and $S_{ij}(t)$, as shown in Figure 2(c):

$$q_{ij}(t) = A_{ij}(t) - S_{ij}(t). \tag{1}$$

The change of flow i's queued bits at node R_j is,

$$\Delta q_{ij}(t) = q_{ij}(t+\Delta t) - q_{ij}(t) = (\overline{\lambda}_{ij}(t,\Delta t) - \overline{\mu}_{ij}(t,\Delta t)) \times \Delta t = I_{ij}(t,\Delta t) - O_{ij}(t,\Delta t) \tag{2}$$

where $I_{ij}(t,\Delta t)$ and $O_{ij}(t,\Delta t)$ are incoming and outgoing bits of flow i at node R_j during the time interval $[t, t+\Delta t]$; $\overline{\lambda}_{ij}(t,\Delta t)$ and $\overline{\mu}_{ij}(t,\Delta t)$ are the correspondent average input and output rates, respectively.

Now consider the flow's queueing behavior at a *sequence* of FIFO nodes. Define flow i's *accumulation as a time-shifted, distributed sum of the queued bits* in all nodes along its path from the ingress node R_1 to the egress node R_J, i.e.,

$$a_i(t) = \sum_{j=1}^{J} q_{ij}(t - \sum_{k=j}^{J-1} d_k) \tag{3}$$

which is shown as the solid slant line in Figure 2(b). Note this definition includes only those bits backlogged inside the node buffers, not those stored on transmission links. With the definitions of

$$\lambda_i(t) = \lambda_{i1}(t),\ \mu_i(t) = \mu_{iJ}(t) \tag{4}$$

[3] This is strictly true if we accept that a bit is infinitely small.

we calculate flow i's accumulation change as follows:

$$\begin{aligned}\Delta a_i(t) &= a_i(t+\Delta t) - a_i(t) \\ &= \sum_{j=1}^{J} \Delta q_{ij}(t - \sum_{k=j}^{J-1} d_k) \\ &= [\overline{\lambda}_i(t - d_i^f, \Delta t) - \overline{\mu}_i(t, \Delta t)] \times \Delta t \\ &= I_i(t - d_i^f, \Delta t) - O_i(t, \Delta t) \qquad (5)\end{aligned}$$

where $d_i^f = \sum_{j=1}^{J-1} d_j$ is the forward direction propagation delay of flow i from node R_1 all the way down to node R_J. Similar to Equation (2), $I_i(t - d_i^f, \Delta t)$ and $O_i(t, \Delta t)$ are flow i's bits coming into and going out of network during two *different* time intervals of length Δt each; while $\overline{\lambda}_i(t - d_i^f, \Delta t)$ and $\overline{\mu}_i(t, \Delta t)$ are the correspondent average ingress and egress rates. The result, illustrated in Figure 2(b), shows the change of a flow's accumulation on its path is only related to its input and output at the ingress and egress nodes. That means it is possible to control accumulation at only the ingress and egress nodes.

Given a time sequence $\{t_1, t_2, \ldots, t_k, \ldots\}$, denote $a_i(t_k)$ as $a_i(k)$, $\overline{\lambda}_i(t_k - d_i^f, \Delta t)$ as $\overline{\lambda}_i(k)$, and $\overline{\mu}_i(t_k, \Delta t)$ as $\overline{\mu}_i(k)$, according to Equation (5), we have,

$$a_i(k+1) = a_i(k) + (\overline{\lambda}_i(k) - \overline{\mu}_i(k))(t_{k+1} - t_k)$$

It shows that accumulation can be measured by using correlated periods, e.g. $[t_k, t_{k+1}]$ at egress and $[t_k - d_i^f, t_{k+1} - d_i^f]$ at ingress. This can be done by sending synchronization data "*out-of-band*," i.e., the synchronization data experience only the fixed one-way delays and not the queueing delays.

Given the fluid flow assumption, denoting the queue length of node j at time $t_k - \sum_{(x=j+1)}^{J} d_x$ as $q_j(k)$, accumulation also satisfies the following property:

$$a_i(k) > 0 \Leftrightarrow \exists j \;\; q_j(k) > 0 \text{ and } a_i(k) = 0 \Leftrightarrow \forall j \;\; q_j(k) = 0 \qquad (6)$$

In other words, for a network of fluid flows *a zero-threshold for the per-loop accumulation measure is equivalent to a zero-threshold on the real queue at some bottleneck*. The properties are rigorously developed in reference[22].

In summary, accumulation and output rate are quantities that can be *measured with only per-flow information.* That allows us to build a *fully* distributed, transparent closed-loop congestion avoidance mechanism. In particular, a simple congestion avoidance approach would be:
a) Use simple thresholding techniques on the accumulation measure to detect epochs of congestion.

b) Use feedback to guide the congestion response policy to achieve fine-grained control over input rate dynamics.

While the above discussion referred to unicast, the same approach can be applied to multicast if the machinery for accumulation measurement can be instrumented, which is the focus of the following sub-sections.

2.2 Accumulation Measurement

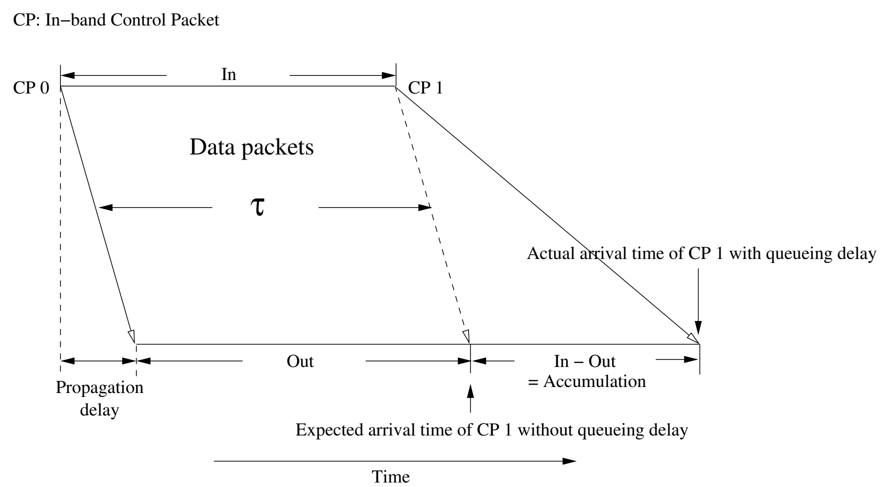


Fig. 3. ACCUMULATION WITH IN-BAND CONTROL PACKETS

To perform end-to-end accumulation measurement in real world, we relax three key assumptions made in the previous section. First, we send synchronization packets (a.k.a control packets)"*in-band*" instead of "*out-of-band*". Therefore, these packets experience both variable queueing delay and fixed propagation delay. Second, we send packets instead of bit-by-bit fluid. Therefore, to account for the randomness introduced, the thresholding procedure has to be amended and a new re-synchronization procedure is performed at the end of each congestion epoch. Third, we develop the scheme for multicast, i.e., we divide functionalities between the source and receivers to minimize the reverse control traffic. Forward control traffic is multicast to all receivers. Note that the first release of assumption is the major reason of approximate accumulation measurement. Again, as we describe in the introduction (Section 1), our aim is to react to congestion as early as possible so as to avoid unnecessary packet loss and increase bottleneck utilization. Therefore, the measurement approximation is acceptable.

Figure 3 illustrates the measurement of accumulation using *in-band* control packets. [4] Assume that the first control packet (CP0) sees no queue and hence arrives at receivers after exact propagation delay. The second control packet

[4] By control packet, we actually mean a data packet with some one-bit flag turned

(CP1) is multicast after the measurement interval τ. A receiver measures *"out"*, the number of bytes received during the period of τ since the receipt of CP0. After CP1 arrives, the receiver knows how much the sender has sent during the period of τ, i.e. *"in"*. $in - out$ is then the accumulation. Observe that this measure works correctly in a rate-based system where packets are sent uniformly and the input burstiness is controlled by a rate-shaper.

Figure 3 still assumed a fluid model. Packetization introduces randomness and burstiness in the system. In particular, even for a perfectly smoothed packetized transmission, an underloaded bottleneck can have an *average* steady state queue of half a packet and a *maximal* queue of one packet for *each flow* going through it. Therefore, the fluid flow formula (6) that implies a zero-threshold on accumulation no longer holds. In our simulation, we use the following hysteresis technique: declare congestion if accumulation becomes larger than two packets, and subsequently declare end of congestion when accumulation falls below one packet. If there are other sources of noise that affects accumulation (eg: scheduling noise at operating systems or at bottlenecks), the thresholds should be set higher. This technique is hence conservative in detecting congestion, i.e., a receiver may unilaterally detect congestion even if there is no network congestion (eg: in multi-bottleneck cases). Higher thresholds reduce the probability of such errors, at the price of larger worst-case queues. We find through simulations that the settings above work very well.

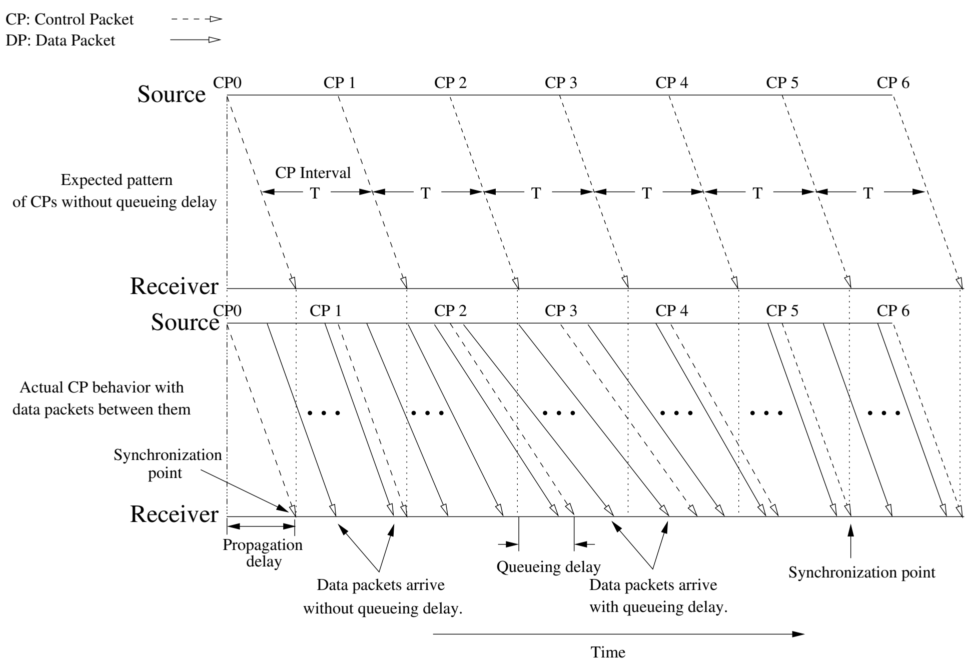


Fig. 4. CONGESTION EPOCHS: SYNCHRONIZATION POINTS AND ACCUMULATION

---

on and with its sending time in the optional field. If this bit is turned off and no sending time is carried, the data packet is then simply a normal data packet.

Again because a bottleneck may have a steady state queue even when underloaded, the initial control packet (CP0 in Figure 3) may not see zero queueing delay. In other words, our assumption of synchronization at the first control packet arrival may be erroneous. Also, as a side effect of the hysteresis scheme described above, a receiver could end a congestion epoch with non-zero accumulation. To counter these issues, we introduce the notion of *"re-synchronization"* as illustrated in Figure 4. We begin with the default assumption that we have synchronized correctly at CP0, and then measure accumulation during successive intervals based upon this assumption. If a control packet arrives at the receiver before its expected arrival time, we re-synchronize (not shown in the figure) and set accumulation to zero. Also, if we detect the end of a congestion epoch, we re-synchronize.[5] Figure 4 shows a case when the re-synchronization happens perfectly, i.e., accumulation is zero, and the re-synchronization point overlaps the expected arrival time of a control packet without queueing delay. In practice, any residual positive accumulation is carried over to the next epoch.

It should be noticed that if the route between source and receiver changes to be longer, re-synchronization requires more effort. There are two possible solutions: (1) The source keeps reducing the sending rate until it reaches zero, and receivers re-synchronize at that point. (2) Out-of-band packets can be used to monitor the shortest end-to-end delay and help adjusting re-synchronization points. However, in our targeted networks mentioned in the introduction, route change rarely happens and thus is ignored in our design.

3 MCA: Scheme Description

First, for clarity, we list the acronyms in the following:

CI stands for *congestion indication*. It is a packet conveyed by receiver to inform the source of congestion.

CR stands for *congestion representative*. It is the slowest receiver. The source makes rate adjustment decisions solely based on its CIs.

G-TRAC stands for *good throughput rate at congestion*. It is defined as the product of receiving rate during congestion epochs and $(1 - f)$, where f is congestion occurrence frequency. It will be explained in more details in Section 3.1.3.

Generally, in MCA, the sender keeps multicasting data and control packets to receivers, and receivers detect congestion upon receipt of control packets by

[5] Believing that the intervals between synchronizations won't be long, we assume that clock skew can be ignored.

measuring accumulation and checking packet loss. When there is not congestion, the sender does not receive any CI, and keeps increasing sending rate periodically. If there is congestion, receivers convey the information to the source by sending CIs, given that these CIs pass a carefully designed suppression filter. When CIs arrive at the sender, another filter is applied to choose the particular CIs from the CR. The sender then reduces the transfer rate based on those chosen CIs.

Therefore, MCA operations can be divided into two categories: those done by the source operations and those by receivers. Details are presented in the following.

3.1 Source Operations

The major task of the source is to adjust sending rate according to the congestion information sent back by receivers. Our approach is to adapt the sending rate to the slowest receiver, called CR in this paper. If there are CIs from the CR, the sending rate is reduced, otherwise the rate is increased once per estimated RTT. Consequently, the source needs to provide solutions to the following problems:

(1) RTT estimation
(2) Rate adaptation
(3) CR switching
(4) CI filtering

Among the solutions, RTT estimation provides estimated RTT for rate adaptation and CI filtering, CI filtering dictates CR switching (i.e. selecting the right CR), and CR switching provides CR for rate adaptation. The key operations are shown in the flow chart of Figure 5, and are explained below.

3.1.1 RTT estimation

RTT is a parameter that affects rate adaptation in several ways as well as CR switching (the effect of RTT on rate adaptation will be presented in next subsection). A sample of RTT is obtained whenever a CI arrives at the source. Its value is the time difference between the CI arrival time and the departure time of the packet triggering the CI. Given a sample value s, the RTT is smoothed using EWMA (exponential weighted moving average), i.e. $SRTT = 7/8 \cdot SRTT + 1/8 \cdot s$. The deviation is calculated as $\sigma = 7/8 \cdot \sigma + 1/8 \cdot (|SRTT - s| - \sigma)$.

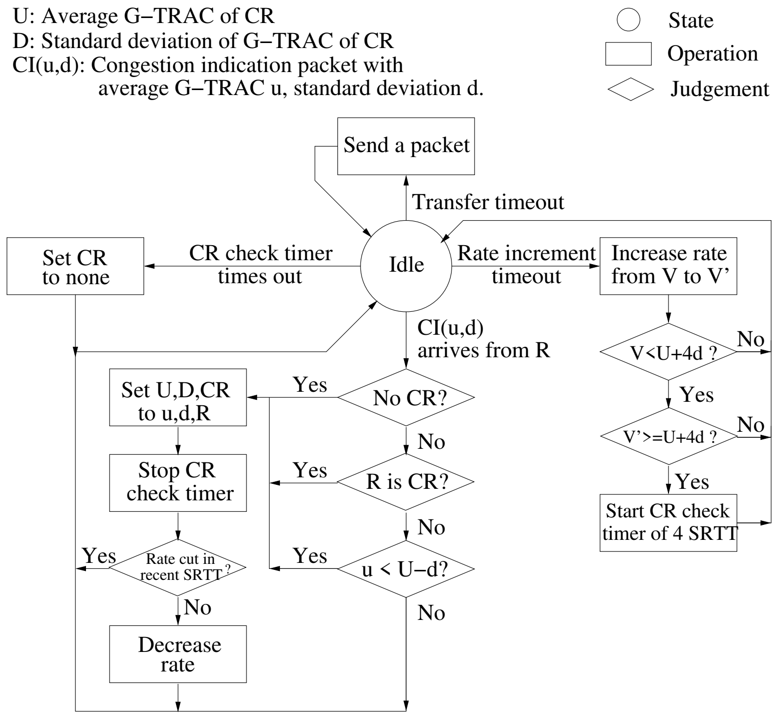


Fig. 5. SOURCE OPERATION FLOW

3.1.2 Rate adaptation

MCA adopts AIMD (additive increase and multiplicative decrease) rate adaptation policy. If no CI arrives, at the end of each period of $SRTT + 4\sigma$, the source increases the transfer rate in order to send one more packet during next period. If a CI arrives and it is from the CR, the source needs to check whether the rate has been reduced during the most recent $SRTT + 4\sigma$. If not, the rate is reduced. This is to guarantee that at most one rate reduction is performed per RTT like TCP does.

In each CI, a single-bit flag indicates whether the CI is triggered by accumulation over threshold or by packet loss. If it is the former, the rate is reduced by 10%; if the latter, the rate is reduced by 25%, since packet loss means more severe congestion. The percentages of cutting rate are subject to heuristic choices. Generally, if they are set higher, congestion will be cleared more quickly while bottleneck utilization may decrease; if set lower, there is more risk of persistent congestion while bottleneck utilization may increase.

3.1.3 CR Switching

Since network condition keeps changing, the choice of CR must be updated accordingly so that the sending rate can always be adapted to the most congested bottleneck. There are mainly two reasons to change CR: (1) The current CR is still active but another receiver becomes the new slowest one, (2) The current CR is absent (e.g. going offline). We have two different techniques to cope with these two situations respectively.

An important metric used in both techniques (as well as receiver-side feedback suppression to be described later) is *Good Throughput Rate At Congestion* (G-TRAC), which is used to measure the congestion level on the path between the source and a receiver. The larger the G-TRAC value is, the lower is the congestion level. G-TRAC is composed of two parts. The first part TRAC is the receiving rate a receiver measures when it detects congestion. To avoid the random error due to burstiness, TRAC is averaged over a short period, for example, the most recent RTT. The second part *congestion occurrence frequency* (f), also measured by receivers, is the number of CIs (either sent or suppressed) divided by the total number of packets sent by the sender over a certain period (e.g the period between two congestion epochs). G-TRAC is then the TRAC weighted by $(1-f)$. Every receiver in a multicast session measures its own G-TRAC and maintains the average and deviation, which are sent in CIs. It should be noticed that congestion occurrence frequency is not equivalent to the commonly known "packet loss probability". There are two reasons: (1) Since MCA is a congestion avoidance scheme, at times of incipient congestion when no packets are lost yet, CIs can still be sent. (2) Even when packets are lost, if several consecutive packets are lost, only *one* CI is sent.

For CR initialization (when there is no CR yet), the source simply chooses the receiver whose CIs arrive at the source first. It will then keep tuning up the choice using the following two techniques.

As shown in Figure 5, when the source receives a CI, it checks the G-TRAC average in the CI. If it is lower than $U-D$ (where U and D are the G-TRAC average and deviation of the current CR respectively), the receiver sending this CI will be chosen as the new CR, and U and D are updated with the values in the CI. We use $U-D$ as the lower bound on the basis of assuming G-TRAC is a random variable. If another receiver has an average G-TRAC lower than $U-D$, it is very likely that the receiver is behind a more congested path than the current CR. By using $U-D$, we are conservative and bias toward the current CR to avoid unnecessary oscillation. Generically, $U-kD$ can be used as the lower bound, where k decides on how quickly CR is updated and thus the level of oscillation. Also, more complicated formulas can be used for comparing G-TRAC's. We choose to use the simple form in order to allow easy implementation for deployment in real networks.

While the source has a choice of CR, it needs to continuously check whether the CR is still alive. The method is, at the moment when the sending rate becomes greater than or equal to $U'+4D'$, the source starts to count. U' and D' are the mean and standard deviation of TRAC (Maintaining these two values does not require much more computation because TRAC is part of G-TRAC). If there is no CI coming from the CR within $n(SRTT + 4\sigma)$ after that, the source deems the CR absent, thus resets the choice of CR, and requests CIs from other receivers. $U' + 4D'$ as the upper bound is used under the assumption that TRAC is a random variable. If a new value of TRAC is larger than $U' + 4D'$, according to Chebychev Inequality, it is very likely that some extraordinary event happens. We then wait for an additional period to further confirm the irregularity. $n = 4$ is the heuristic value used in our simulations. Setting n too large will result in delay of detecting CR absence, while setting n too small can result in erroneous judgments of CR absence.

3.1.4 CI filtering

Basically, only CIs from the CR are accepted. A special situation worth attention is that a CI may trigger CR switching, as discussed in the previous section 3.1.3. If that happens, the choice of CR will be updated. Therefore, that particular CI is now from the CR and hence accepted by the source.

It should be mentioned that most CIs from non-CR receivers have already been suppressed by receivers themselves (to be discussed in Section 3.2.2). Nonetheless, sometimes the source can still receive CIs from multiple receivers during (potential) CR transitions (e.g. several receivers competing for the CR). Therefore, source-side filtering is necessary.

3.2 Receiver Operations

Receivers need to detect congestion and convey the information to the source for it to adjust sending rate. At the same time they should suppress their feedback (CIs) whenever necessary, so that the source won't suffer from feedback implosion. Therefore, two major tasks are performed by receivers,

(1) Congestion detection
(2) Feedback suppression

The operations are explained by Figure 6 and the following specifications.

3.2.1 Congestion detection

Detecting congestion by means of accumulation have been well explained in Section 2.2, so we skip it here. In addition to accumulation, packet loss is also considered in case some extraordinary events occur. That is, whenever packet loss is detected, a CI is also sent. There is an one-bit flag in CI indicating

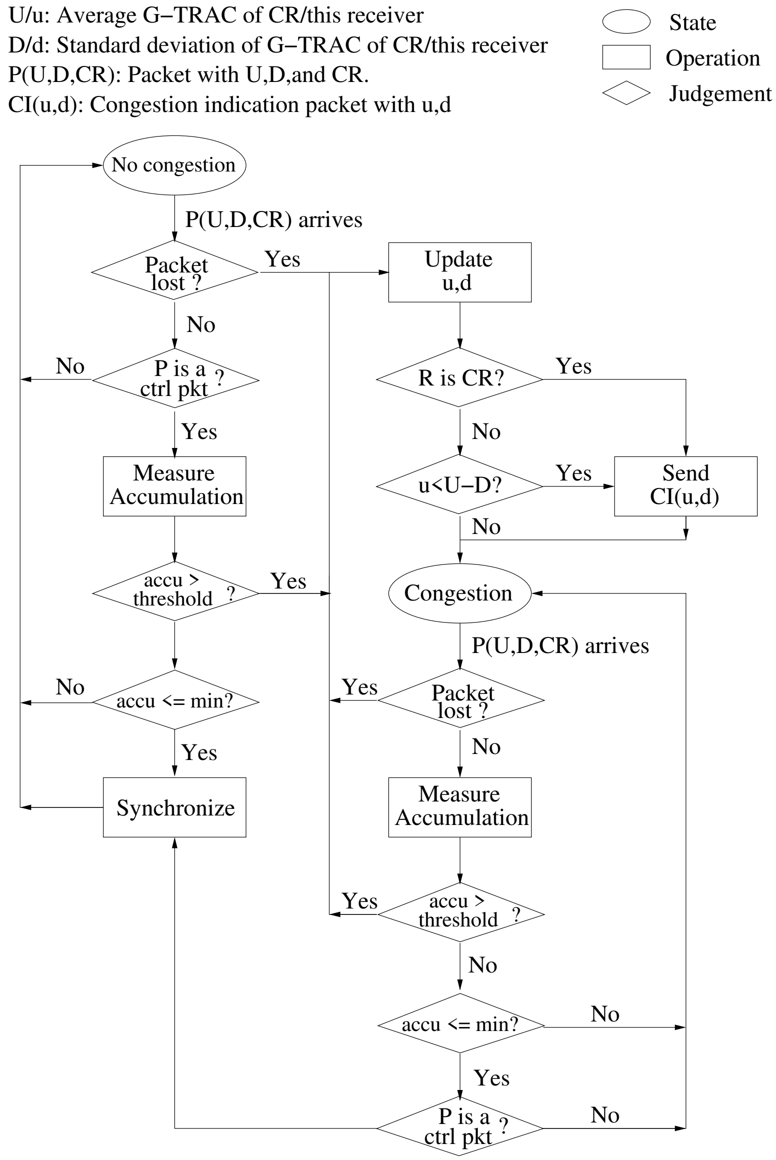


Fig. 6. RECEIVER OPERATION FLOW

whether the congestion is detected by accumulation or by packet loss. The source adjusts the sending rate differently according to this bit (Section 3.1.2).

3.2.2 Feedback suppression

Even if a receiver detects congestion, it does not send CIs if its average G-TRAC (u) is less than $U - D$, where U and D (the G-TRAC average and deviation of the CR) are multicast to receivers by source. If CIs were sent, they would be discarded by the source anyway (Section 3.1.4). The value of $U - D$ is used for the same reason presented in the discussion of CI switching (Section 3.1.3).

By this simple mean, a very large proportion of CIs are suppressed. Note that the feedback suppression here is *non-timer-based*, thus different from those timer-based mechanisms (e.g. the one used in TFMCC [20]). In timer-based feedback suppression mechanisms, feedback packets are scheduled upon detected congestion and then suppressed(canceled) upon packet arrivals from the source. On the contrary, in our scheme, feedback packets are decided to be sent or not *as soon as* congestion is detected. Therefore, no packets are stored in a schedule list.

As we can see, both source operations and receiver operations are simple, and require only small constant number of states. That means MCA is easy to implement and deploy.

4 Simulations

We ran several *ns*-2 simulations to verify the performance of our scheme. The simulations include,

(1) Section 4.1: To verify the correctness of MCA using a simple topology.
(2) Section 4.2: To test the fairness between different MCA sessions in a linear network with multiple bottlenecks.
(3) Section 4.3: To verify that MCA is immune to Drop-to-Zero problem and effective at feedback suppression, and test the fairness between MCA sessions with multiple receivers and those with single receivers.
(4) Section 4.4: To verify that the source of MCA always adapts the sending rate to the most congested bottleneck.
(5) Section 4.5: To test the performance of MCA in a heterogeneous and dynamic network.

In these simulations, the data packet size is 1000 bytes, initial RTT is 0.1 second. We used different bottleneck queue capacities to test MCA performance under situations with or without packet loss. To show the results clearly, we average the sending rates, the utilization rates and queue lengths over one-second periods.

4.1 Basic Test on Simple One-Bottleneck Configuration

We first verify MCA performance on the simple topology in Figure 7. During different periods, there are 10 multicast flows from each source node to all 16 receiver nodes. The flows originated at Src 1 start at the beginning and end

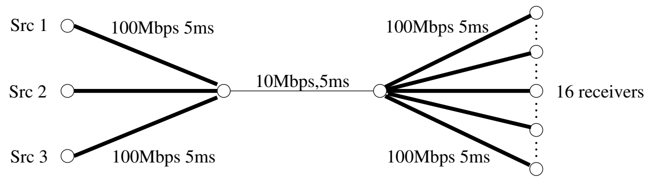


Fig. 7. ONE-BOTTLENECK CONFIGURATION WITH 16 RECEIVER NODES

at 500th second, those originated at Src 2 start at 100th second and end at 400th second, those originated at Src 3 start at 200th second and end at 300th second. Therefore, in different periods, there may be 10, 20 or 30 flows sharing the 10Mbps bottleneck. The simulation time is 500 seconds.

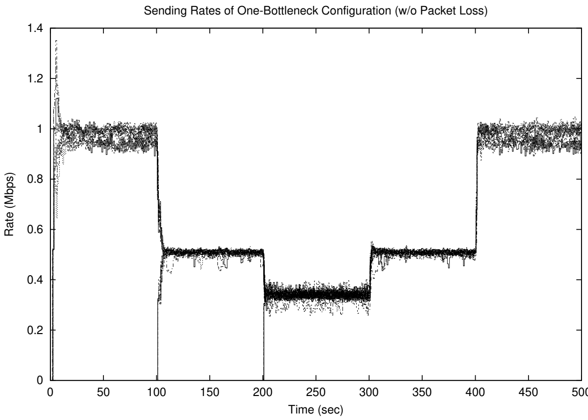
The bottleneck queue capacity was set to 200 packets for lossless situation and 50 packets for lossy situation. As shown in Figure 8, under both situations, the transfer rates adapts to the bottleneck situation, while maintaining high bandwidth utilization and short queue. In reality, bottleneck queue capacity can be much larger than 200 packets. However, in terms of no packet loss, a 200-packet queue is equivalent to any larger queue. On the other hand, the 50-packet queue is just used to guarantee packet loss in order to check the performance of our scheme under extreme situations. The same philosophy applies to the following simulations.

We notice that there are several spikes in the plots of average queue length. That is because ten flows are introduced into the network at those moments. Over a short period, before MCA can react, the total amount of traffic increases significantly, and therefore, more packets are buffered. However, the spikes disappear quickly because congestion is detected and sending rates of all flows are properly adjusted according to MCA.

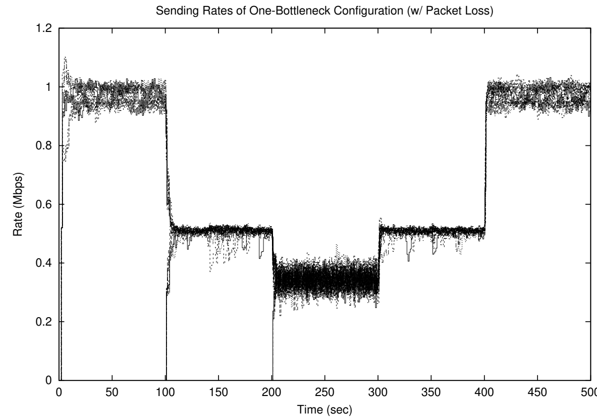
We also notice that the average queue lengths are not zero. As we discussed in Section 2.2, even an underloaded bottleneck can have an *average* steady state queue of half a packet and a *maximal* queue of one packet for *each flow* going through it. Since we have at least ten flows on each bottleneck, it is expected that the average queue lengths are larger than zero. In fact, by doing simple calculation, we can see that the average accumulation (in packets) maintained by each flow is very low. The same explanation applies to the following simulations.

4.2 Fairness Test with Multiple Bottlenecks (Linear Network)

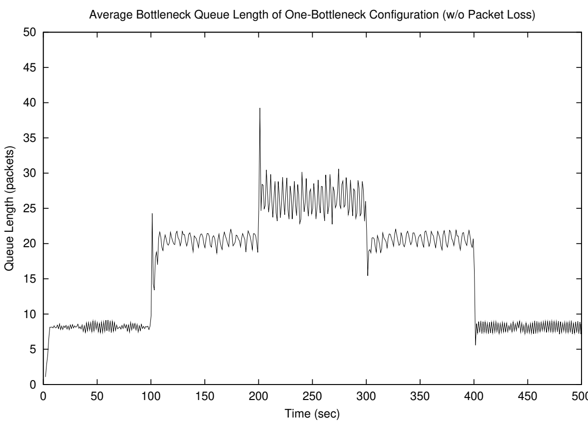
To check how MCA flows compete with each other when they pass different number of bottlenecks and what kind of fairness MCA can achieve, simulations were run on a multiple bottleneck topology (Figure 9).



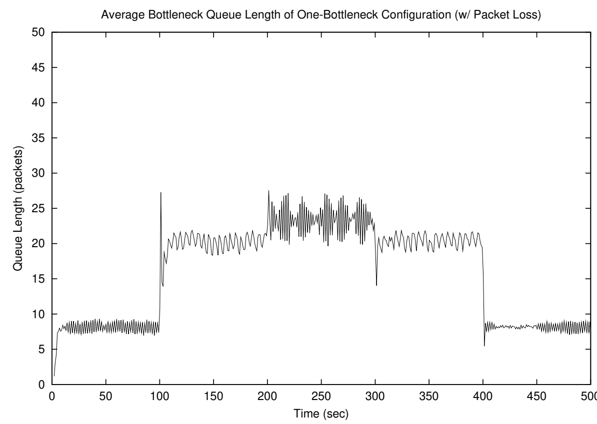
Rate (w/o packet loss)



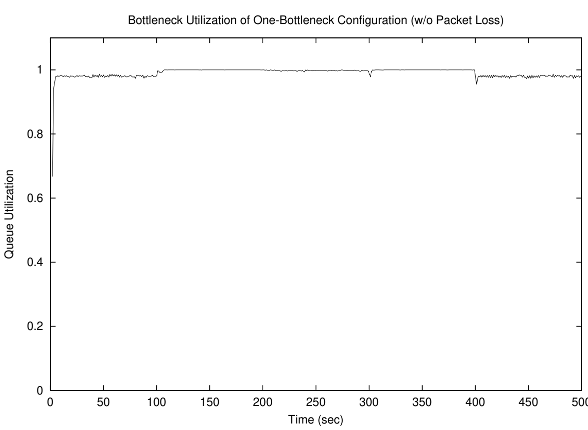
Rate (w/ packet loss)



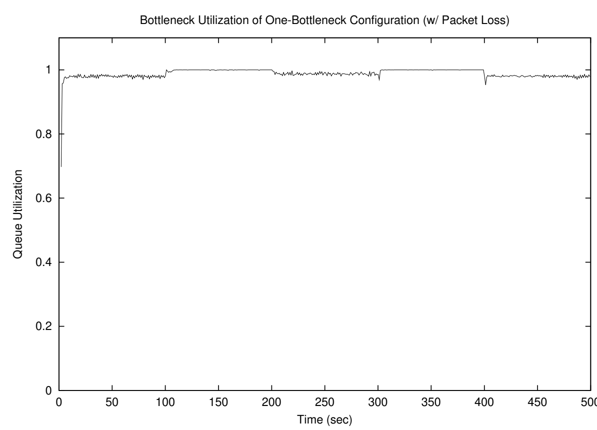
Average Queue Length (w/o packet loss)



Average Queue Length (w/ packet loss)



Bottleneck Utilization (w/o packet loss)



Bottleneck Utilization (w/ packet loss)

Fig. 8. IN ONE-BOTTLENECK CONFIGURATON, MCA ADAPTS TO BOTTLENECK CHANGE WITH HIGH BANDWIDTH UTILIZATION AND LOW AVERAGE QUEUE LENGTH.

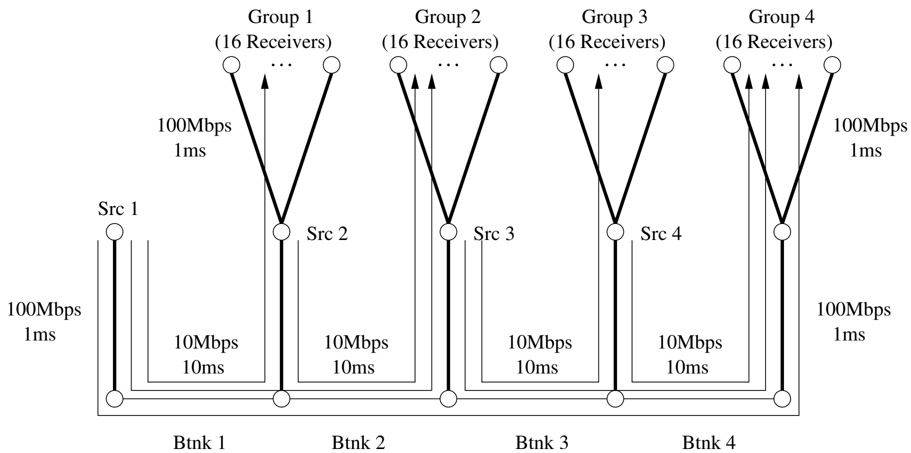


Fig. 9. LINEAR NETWORK: MULTIPLE BOTTLENECKS CONFIGURATION

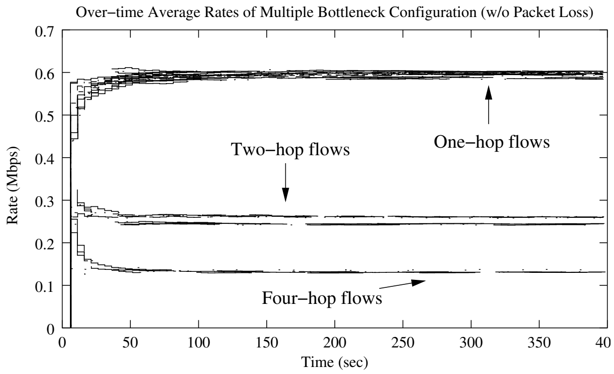
In this topology, there are four 10Mbps bottlenecks. Other links are 100Mbps. To reduce the effect of RTT, we set the bottleneck delays to 10 milliseconds and others to 1 millisecond. There are three types of flows: one-hop flows (i.e. flows going through one bottleneck), two-hop flows and four-hop flows, as shown in Figure 9. Among them, for $i = (1, 2, 3, 4)$, 10 one-hop flows start at Src i and end at all the 16 receivers in Group i; for $i = (1, 3)$, 10 two-hop flows start at Src i and end at Group $i + 1$; and there are 10 one-hop flows going from Src 1 to Group 4. Therefore, each bottleneck is shared by 30 flows. Bottleneck buffer size is set to 200 and 40 packets for lossless and lossy situation respectively. The simulation time is 400 seconds.

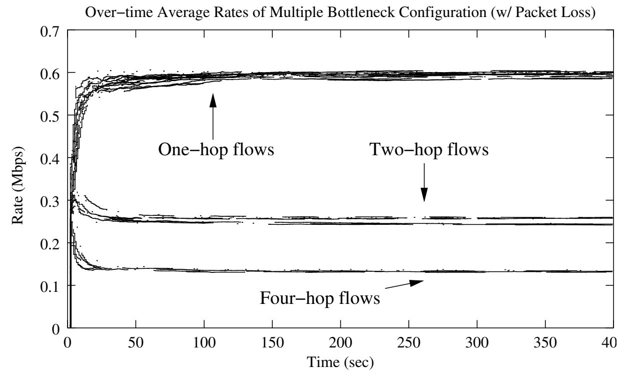
The average rates [6] in Figure 10 show that the one-hop flows used approximately 2.5 times as much bandwidth as that by two-hop flows, while the two-hop flows used almost twice as much bandwidth as that by four-hop flows, which is close to what proportional fairness suggests theoretically. Again, the bottleneck queue length is low, and the utilization is high, confirming the good performance of MCA. [7]

4.3 Test of Drop-to-Zero Avoidance, Friendliness to Unicast Flow and Feedback Suppression

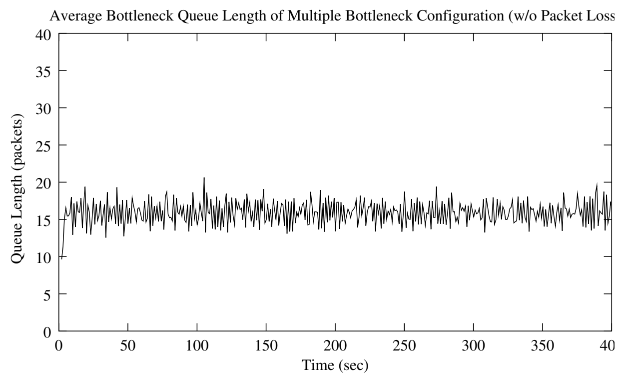
It is critical for a multicast congestion avoidance/control scheme to avoid reacting to more feedback than necessary, otherwise the source will reduce the sending rate too often and therefore keep it very low or even zero, which is known as the Drop-to-Zero problem. We designed a star topology in Figure 11

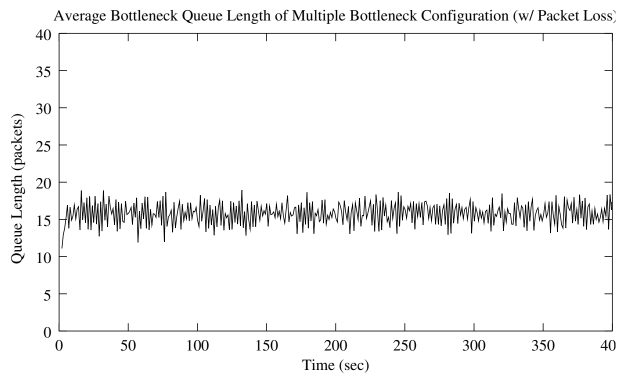
[6] Average rate at time t is defined at the amount of data sent during $[0, t]$ divided by t.
[7] The results are of one randomly chosen bottlenecks. The situations of other bottlenecks are similar.



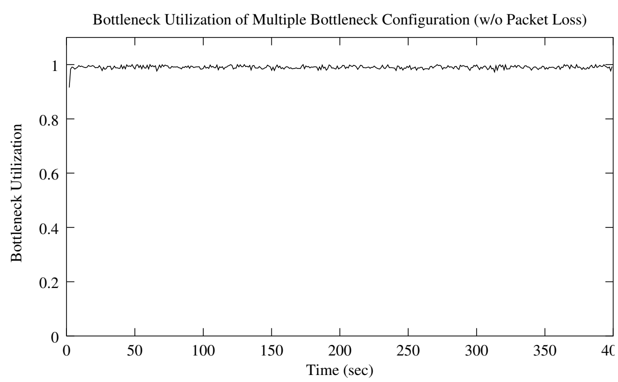


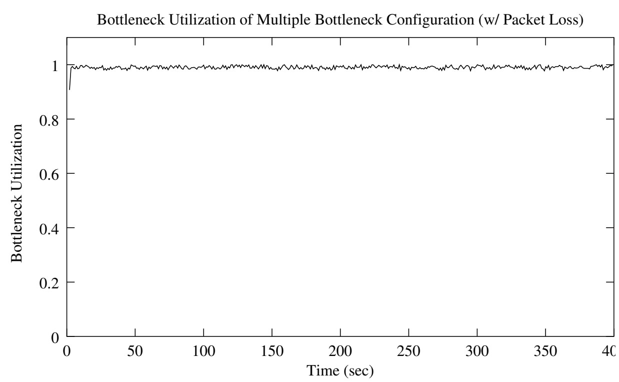
Rate (w/o packet loss) Rate (w/ packet loss)





Average Queue Length (w/o packet loss) Average Queue Length (w/ packet loss)





Bottleneck Utilization (w/o packet loss) Bottleneck Utilization (w/ packet loss)

Fig. 10. IN MULTIPLE BOTTLENECK CONFIGURATON, MCA ACHIEVES APPROXIMATELY PROPORTIONAL FAIRNESS.

to generate asynchronous congestion on 64 different bottlenecks and checked the performance of MCA. Bottlenecks of 1Mbps bandwidth and 5 millisecond delay are the links between the router and receivers. Between each pair of Source i and Receiver i ($i = 1 \ldots 64$), there are three unicast MCA flows (i.e. MCA flows with only one receiver). Also, there is a multicast MCA flow going from Source 65 to all 64 receivers. In consequence, each bottleneck is shared by four flows, which congest the link in an asynchronous manner. Bottleneck buffer size is set to 200 and 10 packets for lossless and lossy situation

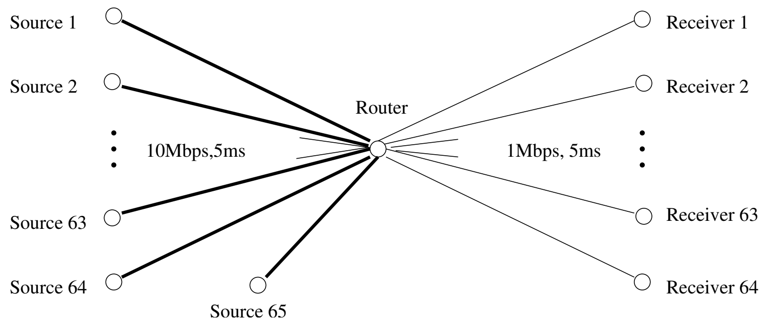


Fig. 11. 64-RECEIVER STAR TOPOLOGY

respectively. The simulation time is 400 seconds.

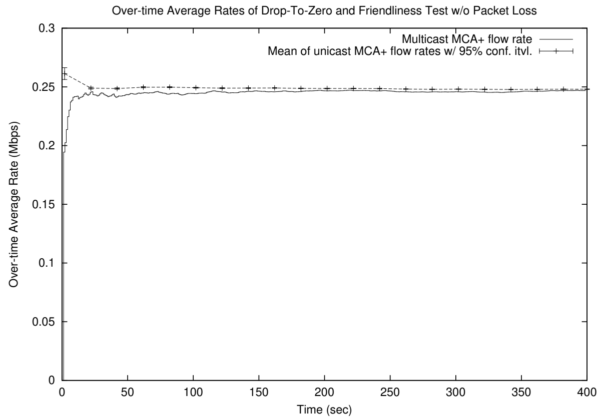
The over-time average sending rates (the mean and confidence interval of unicast flow rate are calculated with the samples of all 192 unicast flows) in Figure 12 show that the throughputs of the multicast MCA flow and the unicast flows are almost the same, no matter there is packet loss or not. The high bottleneck utilization and low average queue length shown in the figures are of one of the bottlenecks. Results of other bottlenecks are very similar.

In this simulation, under the lossy situation, the total number of feedback packets (CIs) sent by the multicast receivers is 5444, while the average number of CIs which would have been sent per receiver without feedback suppression is 4978, and the average number of CIs sent by unicast flow receivers is 5057. We can see that the amount of CIs sent in the multicast session is very close to that by a unicast receiver, which indicates that *the feedback suppression mechanism in MCA is highly efficient.* For reference, under the lossless situation, the three numbers are 5605, 5135 and 5147 respectively, again close to each other.

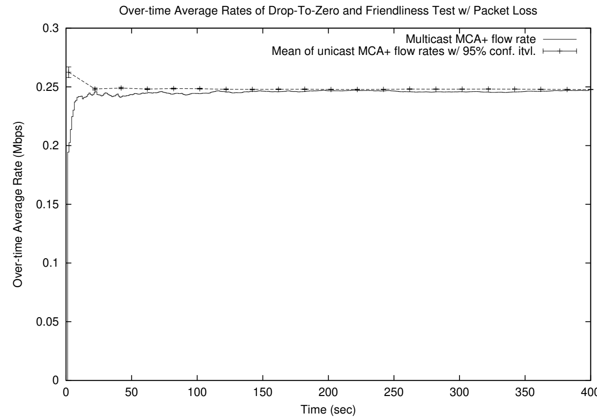
4.4 Test of Tracking The Most Congested Bottleneck

In this simulation, we changed the most congested bottleneck in a pre-defined pattern, and check whether the choice of CR can be correctly updated as the most congested bottleneck changes. There is a multicast MCA flow from the source to all 32 receivers (Figure 13). During the whole simulation, one unicast MCA flow exists between the source and each receiver. At 200th, 400th and 600th second, we introduce 2, 3, 4 unicast MCA flows to the links between the source and Receiver 2, 3, 4 respectively. At 800th, 1000th and 1200th second, we stop the added flows in the reverse order. As the result, the most congested bottlenecks during different periods are as shown in Table 1. The bottleneck buffer size is set to 200 packets and 8 packets respectively for lossless and lossy situations.

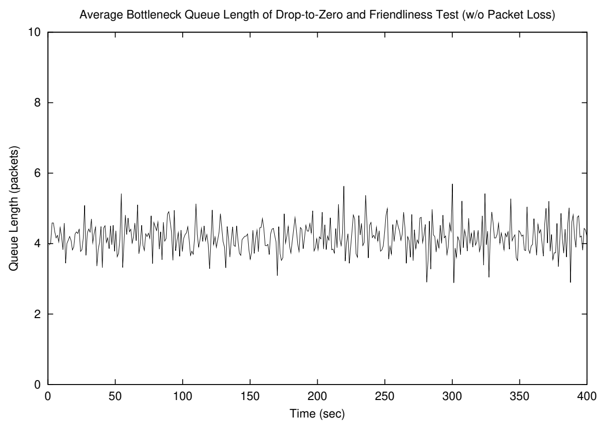
Rate changes plotted in Figure 14 indicate that the multicast MCA flow always



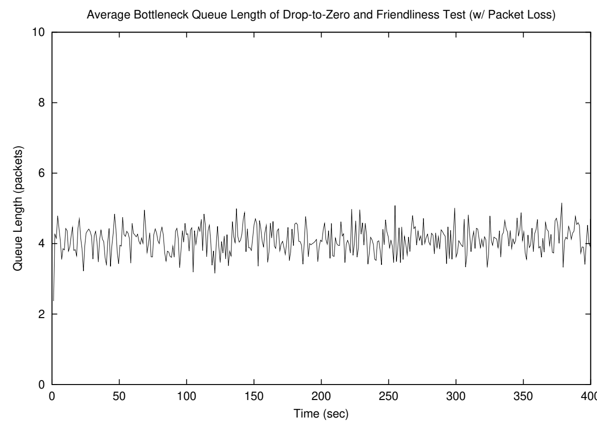
Rate (w/o packet loss)



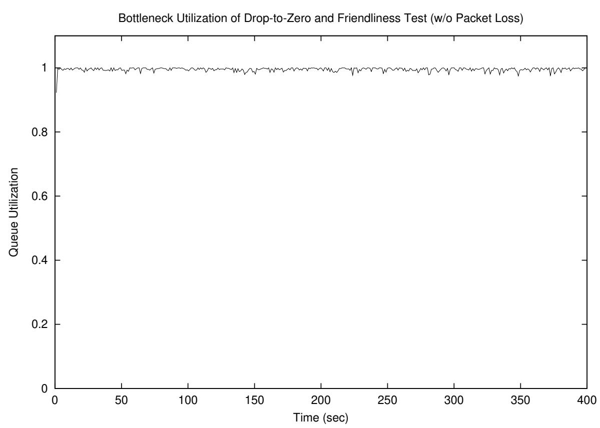
Rate (w/ packet loss)



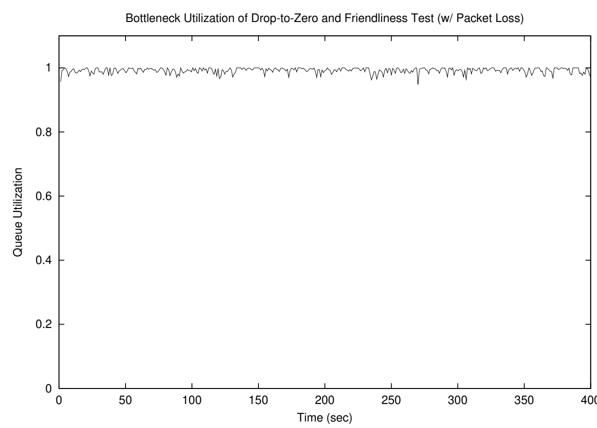
Average Queue Length (w/o packet loss)



Average Queue Length (w/ packet loss)



Bottleneck Utilization (w/o packet loss)



Bottleneck Utilization (w/ packet loss)

Fig. 12. MULTICAST MCA FLOW GETS APPROXIMATELY THE SAME THROUGHPUT AS UNICAST FLOWS DO. THEREFORE, MCA IS IMMUNE TO DROP-TO-ZERO PROBLEM AND IS FRIENDLY TO UNICAST FLOWS.

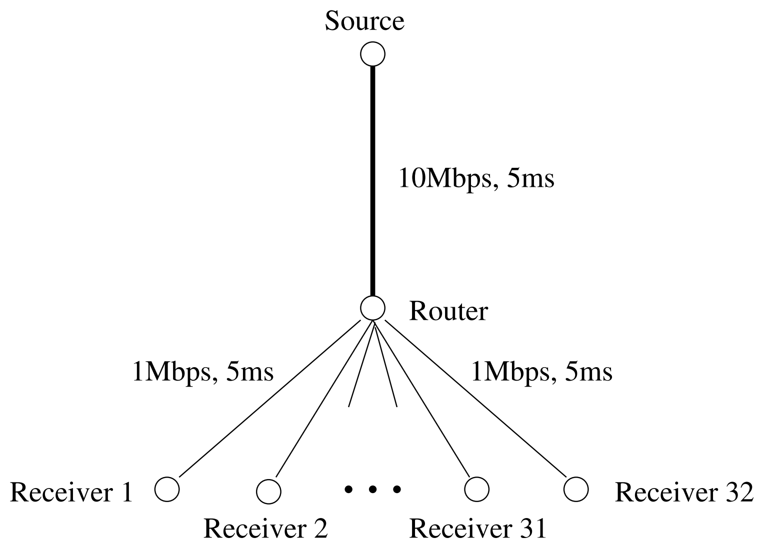


Fig. 13. 32-RECEIVER TREE TOPOLOGY

Table 1
DYNAMICS OF MOST CONGESTED BOTTLENECK

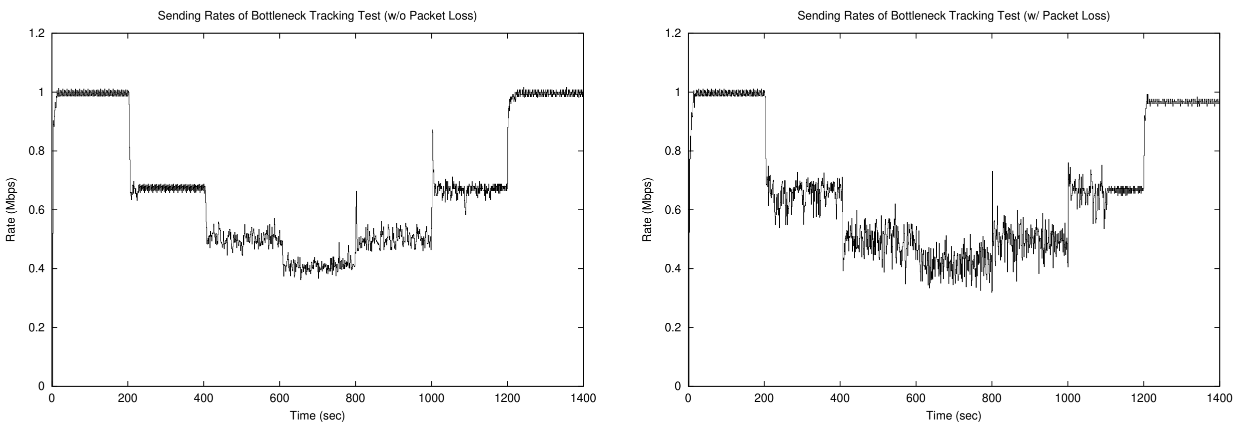
Periods	Most Congested Link
[0, 200) and [1200, 1400]	Link 1
[200, 400) and [1000, 1200)	Link 2
[400, 600) and [800, 1000)	Link 3
[600, 800)	Link 4

(Link i is the link between Router and Receiver i.)

tracked the most congested bottleneck, showing the effectiveness of mechanisms that control CR switching. There are more oscillations between 400th and 1000th second than other time. It is because within that period the situations of the most congested bottleneck and the not-so-congested bottlenecks are close, and there are some back and forth CR switching. We focus on CR switching in this simulation, therefore we don't show the bottleneck utilization and queue length figures.

4.5 Test of Performance in Dynamic Network

It is also desirable to test the performance of MCA with *random* traffic patterns. We designed a network with heterogeneous delays, as shown in (Figure 15). Each link has 2Mbps bandwidth. Among all the links, 2 links at the first level, 4 links at the second level, and 8 links at the third level have 200ms delay, while all other links have 20ms delay. We arrange the links so that on any path between the source and a receiver, there is at most one link of



Rate (w/o packet loss) Rate (w/ packet loss)

Fig. 14. THE SENDING RATE OF MCA ALWAYS ADAPTS TO THE MOST CONGESTED BOTTLENECK.

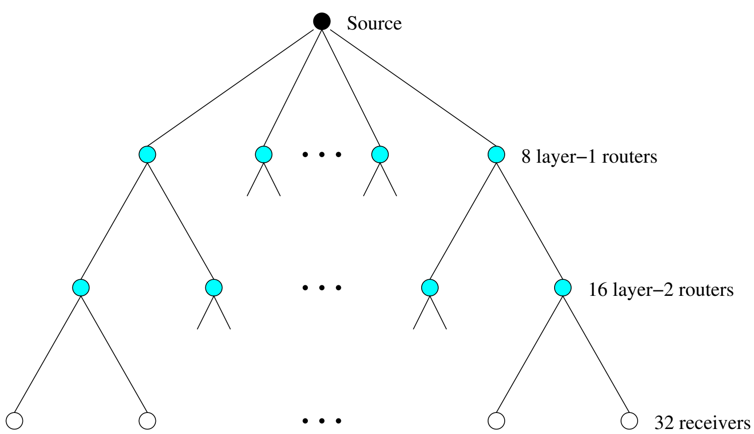


Fig. 15. HETEROGENEOUS DYNAMIC NETWORK

200ms delay. Furthermore, we generate dynamic traffic in this network. On each link, three unicast MCA flows are randomly turned on and off according to Pareto distribution with average period length of 60 and 30 seconds respectively. Moreover, all receivers except one in the multicast session join and leave randomly, again according to Pareto distribution. The average in-session time is 60 seconds and average out-of-session time is 30 seconds. We keep one receiver always in the session so that feedback always exists and the sending rate won't increase infinitely. Bottleneck buffer sizes are set to 50 packets.

We ran the simulation for ten times. The average throughput of the multicast MCA flow is **232** Kbps, with standard deviation of **10** Kbps, which shows that MCA works well in a heterogeneous and dynamic environment. The average number of total feedback packets received by the multicast source is 3009, with standard deviation of 777, again showing the effectiveness of feedback suppression.

5 Conclusion

We have proposed MCA, an end-to-end rate-based multicast congestion *avoidance* scheme. This scheme leverages the concept of *accumulation* (the number of buffered bits of a flow inside the network) developed in our recent work [22]. Using accumulation measurement extended to multicast, receivers detect congestion without necessarily inducing packet loss. Upon congestion, receivers send congestion indications (CIs) back to source for the purpose of rate control. *Non-timer-based* feedback suppression is invoked before CI sending. At the other side, the source keeps a record of the slowest receiver as congestion representative (CR), and only accepts its CIs for adapting the transfer rate according to AIMD rate control policy. Both CR switching and feedback suppression make use of a new metric, *Good Throughput Rate At Congestion* (G-TRAC) (defined as the product of receiving rate during congestion epochs and $1 - f$, where f is congestion occurrence frequency). Receivers do not need to *continuously* exchange packets with the source. Feedback implosion is therefore avoided.

We evaluate MCA with detailed simulations in ns-2. Regardless of bottleneck buffer size (that may or may not be large enough to prevent packet loss), the scheme (1) does not suffer from the Drop-to-Zero problem, and, (2) is friendly to unicast flows, (3) achieves high bottleneck utilization and low average queues, (4) is approximately proportionally fair. Due to its congestion avoidance nature, MCA is not compatible with traditional TCP doing congestion control. Therefore, we expect it to be deployed in networks under full control of their administration authorities where multicast efficiency is desired. The authorities can introduce congestion avoidance flows by configuring routers to separate them from congestion control flows. Campus networks or intranets are in the scope. Internet2 is also a possibility because routers in it usually support multicast and are more open to configuration than those in the traditional Internet.

References

[1] S. Bhattacharyya, D. Towsley and J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," *INFOCOM*, March 1999.

[2] S. Bradner et al, "IETF criteria for evaluating reliable multicast transport and application protocols," *RFC 2357*, June 1998.

[3] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE JSAC,* Vol 13, No. 8, October 1995.

[4] J. Byers, M. Luby, M. Mitzenmacher, "Fine-Grained Layered Multicast", *INFOCOM*, April 2001.

[5] J. Byers, G. Kwon, "STAIR: Practical AIMD Multirate Multicast Congestion Control", *NGC*, November 2001.

[6] J.W. Byers, et al, "FLID-DL Congestion Control for Layered Multicast," *NGC*, November 2000.

[7] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN*, Vol. 17, No. 1, June 1989, pp. 1-14.

[8] R. Cruz, "Quality of Service Guarantees in Virtual Circuit Switched Networks," *IEEE Journal on Selected Areas in Communications*, 13(6):1048-1056, August 1995.

[9] Dante DeLucia, Katia Obraczka, "A Multicast Congestion Control Mechanism for Reliable Multicast", *Proceedings of the IEEE ISCC'98*, August 1997.

[10] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *RFC 2460*, December 1998.

[11] S.H. Low, "A Duality Model of TCP and Queue Management Algorithms," *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, September 2000.

[12] S.H. Low, L.L. Peterson, and L. Wang, "Understanding Vegas: A Duality Model," *Proceedings of ACM SIGMETRICS*, June 2001.

[13] J. Mo, R. La, V. Anantharam and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," *Proc. INFOCOM'99*, March 1999.

[14] N. Natu, P. Rajagopal, S. Kalyanaraman, "GSC: A Generic Source-based Congestion Control Algorithm for Reliable Multicast," *Journal of Computer Communications*, Vol 24, No. 5-6, pp. 575-589, March 2001.

[15] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. on Networking*, 1(3):344–357, June 1993.

[16] Jon Postel, "INTERNET PROTOCOL," *RFC 791*, September 1981.

[17] K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," *RFC 2481*, January 1999.

[18] L. Rizzo, "PGMCC: A TCP-friendly Single-Rate Multicast Congestion Control Scheme," *SIGCOMM*, August 2000.

[19] Puneet Thapliyal, Sidhartha, Jiang Li, Shivkumar Kalyanaraman, "LE-SBCC: Loss-Event Oriented Source-Based Multicast Congestion Control", *Multimedia Tools and Applications*, Vol. 17, No. 2-3, pp. 257-294, July - August 2002.

[20] Jorg Widmer, Mark Handley, “Extending Equation-based Congestion Control to Multicast Applications”, *SIGCOMM 2001*, August 2001.

[21] L. Vicisano, L. Rizzo and J. Crowcroft, “TCP-like congestion control for layered multicast data transfer,” *INFOCOM*, April 1998.

[22] Y. Xia, et. al, “Accumulation-based Congestion Control,”, submitted work, 2002. Available at http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html.