Explicit Rate Multicast Congestion Control ⋆

Jiang Li, [b] Murat Yuksel, [a,*] Shivkumar Kalyanaraman [a]

[a] *Rensselaer Polytechnic Institute, ECSE Department, 110 8th Street, Troy, NY 12180, USA.*

[b] *Howard University, Systems and Computer Science Department, 2300 6th Avenue, N.W. Washington, DC 20059, USA.*

Abstract

In this article, we propose a new single-rate multicast congestion control scheme called *Explicit Rate Multicast Congestion Control (ERMCC)* based on a new metric, *TRAC (Throughput Rate At Congestion)*. ERMCC achieves an $O(1)$ memory complexity to maintain state information at source and receivers; requires only simple computations; and does not necessitate measurement of RTTs from all receivers to the source. Furthermore, ERMCC is TCP-friendly, does not suffer from the *drop-to-zero* problem, and is very effective with feedback suppression. Theoretical analysis of the scheme performance is provided, and simulations have shown that ERMCC outperforms PGMCC and TFMCC under most situations. We have also implemented ERMCC over UDP and successfully run it on real testbed systems in Emulab with very good results. In addition to this implementation, we also obtained very good results in large-scale simulation tests of ERMCC.

Key words: Multicast, congestion control, single-rate, drop-to-zero, TCP friendliness, feedback suppression, throughput rate at congestion (TRAC)

⋆ This work was supported in part by NSF Contract AN19819112, ARO Contract DAAD19-00-1-0559 and grants from Intel and Reuters.

* Corresponding author. Email: yuksem@ecse.rpi.edu, Address: RPI, ECSE Department, JEC 6049, 110 8th Street, Troy, NY 12180, USA. Phone: +1 (518) 276 6823, Fax: +1 (925) 888 2167

Email addresses: `lij@scs.howard.edu` (Jiang Li,), `yuksem@ecse.rpi.edu` (Murat Yuksel,), `shivkuma@ecse.rpi.edu` (Shivkumar Kalyanaraman).

1 Introduction

IP multicast is efficient for transmitting bulk data to multiple receivers. There are two categories of multicast congestion control. One of them is single-rate, in which the source controls the data transmission rate and all receivers receive data at the same rate. The previous work includes, for example, DeLucia et. al's work in [1], PGMCC[2], TFMCC[3], MDP-CC[4] and our prior work LE-SBCC [5]. The other is multi-rate (a.k.a layered multicast congestion control), in which receivers join just enough layers in the form of multicast groups to retrieve data as fast as they can. The most noticeable among them are recently developed Fine-Grained Layered Multicast [6] and STAIR [7].

The single-rate category is easy to implement and deploy, because it does not require support from intermediate nodes beyond standard multicast capabilities, also does not introduce high processing load to them. Although such schemes do not scale as well as multi-rate ones because they track the slowest receiver, they are suitable for such situations as the multicast in a not-so-heterogeneous environment, or bulk data transfer without concerns over delay. With some network support [8], we can also emulate multi-rate schemes by deploying single-rate schemes on selected intermediate nodes.

In this paper, we introduce a new single-rate multicast congestion control scheme Explicit Rate Multicast Congestion Control (ERMCC), and show its superior performance under most of the realistic conditions. We will first very briefly describe ERMCC below, and then in Section 2, we will briefly discuss some related work followed by the ERMCC details in Section 3. In Section 4 we will provide theoretical analysis of ERMCC where some proofs will be left to Appendices. Then, we will present simulation and experiment results in Section 5. Finally, we will conclude the paper in Section 6.

1.1 Brief Description of ERMCC

The key idea of ERMCC is to base the scheme on a new metric, *TRAC (Throughput Rate At Congestion)*, which is the throughput rate measured by receivers when congestion is detected. ERMCC is ***practical*** because, (i) at the source and receivers, $O(1)$ state is maintained, and only simple computations are required; (ii) there is no need to measure RTTs from all receivers to the source, which can be a tedious problem especially without external instrumentation (e.g. GPS, NTP server), and (iii) we do not make any assumption on network topology and intermediate nodes beyond standard multicast capabilities. It is also ***effective*** because (1) it successfully addresses the well-known problems of slowest receiver tracking, TCP-friendliness, and drop-to-zero; and

(2) the feedback suppression mechanism works very effectively by suppressing over 95% feedback under normal situations. In fact, it outperforms PGMCC[2] and TFMCC[3] under most situations.

The general concept of our scheme is as follows: The source dynamically selects one of the slowest receivers as *Congestion Representative (CR)*, and only considers its feedback for rate adaptation. The slowest receivers are those with the lowest average TRACs. Each receiver keeps measuring its TRAC when it detects congestion and updates its average TRAC by means of a smoothing technique such as Exponentially Weighted Moving Average (EWMA). Receivers detect congestion, when they observe a loss in the data packets [1]. The source considers these average TRACs of the slowest receivers in its decision to select the CR. When there is no CR, all receivers may send feedbacks to the source. However, this no-CR situation will last at most one RTT, because the new CR will be chosen in one RTT. This limitation of one RTT time period on no-CR case also prevents any possible ack implosion. Once a CR is selected, only the CR and those receivers with average TRAC lower than that of the CR can send feedbacks so that feedbacks are efficiently suppressed. Also notice that our scheme is not concerned with reliability issue and only considers congestion control. Therefore, it is applicable to both reliable and unreliable multicast.

An example operation can illustrate how our scheme works more clearly. In Figure 1 (a), let's assume that at time t_0 the source has chosen a receiver behind the most congested path as CR by comparing average TRACs of receivers. Only the CR will send feedback while other receivers suppress their feedback. These feedbacks are indeed *congestion indication*s (CIs), because they are sent only when congestion is detected due to packet loss. As shown in Figure 1, the feedback from the current CR is the *average TRAC* $\hat{\mu}(t_0) = \Phi(\hat{\Omega}(t_0), \alpha)$, where $\hat{\Omega}(t_0)$ is the *TRAC* of the current CR at time t_0 and $\Phi()$ is an EWMA averaging function with α being the exponential averaging factor, which will later be defined in detail. In addition, the TRAC, $\hat{\Omega}(t_0)$, for the current CR is calculated by averaging *instantaneous output rate* $\hat{\omega}(t_0)$ of the current CR over a small period of time. [2]

Assume that, after some time another path becomes the new most congested path. After a while at time t_1, those receivers $1..k$ behind that path will see average TRACs lower than that of the current CR (i.e. $\forall i = 1..k, \mu_i(t_1) <$

[1] Note that it is also possible to use additional techniques to detect congestion. We do not focus on this to assure needed emphasis on the multicast congestion control rather than congestion detection.

[2] These definitions of the different kinds of TRACs correspond to averaging at two different time-scales with two different methods, and they are calculated in the same manner for all receivers. We will give more detailed explanation of these definitions later.

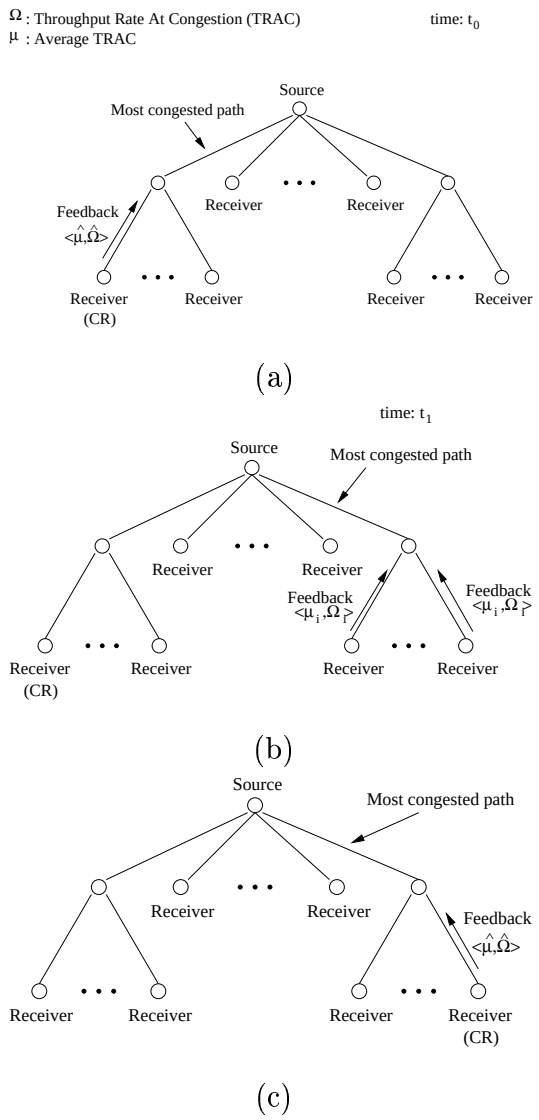


Fig. 1. Example operation of ERMCC.

$\hat{\mu}(t_0) - \hat{\sigma}(t_0))$, and will send feedbacks as shown in Figure 1 (b). As the result, one of them will be chosen as the new CR. After that, again, other receivers will suppress their feedback as shown in Figure 1 (c).

1.2 Key Contributions

ERMCC introduces a novel method of using *explicit rate* feedback at the time of congestion (i.e. TRAC) in such a way that several major multicast congestion control problems are remedied. By using *smoothing techniques* like EWMA, receivers in ERMCC successfully achieve an *efficient feedback suppression*. Similarly, each receiver maintains two *statistical measures* (i.e. average TRAC and deviation of TRAC) which provides venue for *robust and*

effective tracking of the slowest receiver. By using an AIMD-like rate adaptation technique, ERMCC also warrants *TCP-friendliness* and immunity to *the drop-to-zero problem*. In addition, only state of the slowest receiver (i.e. $O(1)$ memory complexity) is needed and only estimation of the RTT to the CR is needed.

The contributions of this paper are as follows:

- It proposes ERMCC, a practical and effective single-rate multicast congestion control scheme based on a new metric *throughput rate at congestion* (TRAC), with features such as $O(1)$ state complexity, and *non-timer-based* feedback suppression.
- It analyzes the scheme performance theoretically.
- By comparison in simulation with PGMCC [2] and TFMCC [3], it shows that ERMCC achieves better performance under most situations.
- It presents the test results of ERMCC implementation on a real testbed system in Emulab [9].
- It also presents large-scale simulation test results of ERMCC.

2 Related Work

2.1 Single-Rate Schemes

DeLucia et. al's work in [1] is an early single-rate multicast congestion control scheme using representatives. It requires two types of feedback from receivers, *Congestion Clear (CC)* and *Congestion Indication (CI)*. Note that their CIs are single bit and thus different from ours carrying the explicit output rate μ. A fixed number of receiver representatives are maintained at the source. Whenever a CI is received by the source, if the sender of this CI is in the representative set, the representative is refreshed; if not, the sender will replace the representative that has not been refreshed for the longest time. Feedback from representatives is echoed by the source to suppress feedback scheduled at non-representative receivers. The source uses only the feedback from representatives to do MIMD (multiplicative increase and multiplicative decrease) rate adaptation.

The representative selection mechanism in that scheme is "simplistic"[1], but there is certain complexity involved in generating CC. The representative set is not guaranteed to include the slowest receiver, which means that the slowest receiver can be overloaded. Furthermore, it assumes that only a few bottlenecks cause most of the congestion. Based on this assumption, receiver suppression is the only mechanism for filtering feedback from receivers. In a heterogeneous

Table 1
MATHEMATICAL NOTATIONS

Symbol	*Meaning*
$\mu_i(t)$	Average TRAC for receiver i at time t
$\Omega_i(t)$	TRAC for receiver i at time t
$\omega_i(t)$	Instantaneous output rate at receiver i at time t
$\sigma_i(t)$	Deviation of TRAC at receiver i at time t
$\hat{\mu}(t)$	Average TRAC for the current CR at time t
$\hat{\Omega}(t)$	TRAC for the current CR at time t
$\hat{\omega}(t)$	Instantaneous output rate at the current CR at time t
$\hat{\sigma}(t)$	Deviation of TRAC at the current CR at time t
$\Phi()$	EWMA averaging function
α	Exponential averaging factor
T	Response time of the CR, i.e. elapsed time until the first feedback is received from the current CR when the path to CR is fully loaded
T_{max}	Estimate of the maximum possible T
$\lambda(t)$	Source's sending rate at time t
s	Data packet size
Δt	Time period over which TRAC is measured
RTT_{max}	Maximum RTT observed by the source among all receivers

network, where there may be many different bottlenecks and asynchronous congestion, the assumption may not be true. Consequently, the transmission rate may be reduced more than necessarily and stay very low or close to zero. This is known as the *drop-to-zero* problem.

PGMCC [2], TFMCC [3] and MDP-CC [4] are recent work also using representatives. Although they use different policies for rate adaptation, they all leverage the TCP throughput formula [10] [11] for allocating the slowest receiver, i.e the receiver with the lowest estimate TCP throughput according to the formula. Therefore, it is necessary for them to measure packet loss rate and RTT for all receivers.

PGMCC [2] keeps one representative as *acker*. The acker sends ACKs to the source which mimics the behavior of TCP. At the same time, NAKs with loss rate are sent from all other receivers. This is different from our scheme because we do not require separate ACK streams. The PGMCC source measures RTT between itself and all receivers in terms of packet numbers, and

compare the estimated throughput for updating acker. Due to the necessity of RTT measurement for all receivers, feedback suppression may have serious effect on PGMCC's performance. In fact, PGMCC does not provide a feedback suppression mechanism.

TFMCC [3] adjusts the rate according to the estimated rate calculated by the representative. RTTs are measured by receivers with a somewhat complex procedure. The sender needs to echo receiver's feedback according to some priority order, and there is one-way delay RTT adjustment plus sender-sider RTT measurement. TFMCC comes with feedback suppression which is an enhanced version of [12] and is probabilistic timer-based. Therefore, the total number of feedbacks is the function of the estimated total number of receivers, and additional delay is introduced into feedback.

MDP-CC [4] increases/decreases the transmission rate exponentially toward the target rate. Similar to TFMCC, the target rate is also calculated by the representative. In contrast to PGMCC and TFMCC, MDP-CC maintains a pool of representative candidates for representative update. As shown in that paper, maintaining multiple representative candidates requires much effort. MDP-CC can use probabilistic timer-based feedback suppression which has the same properties as that of TFMCC.

LE-SBCC [5] is our prior work. It only requires single bit NAKs from receivers, and the source has three cascaded filters to filter receiver feedback before using it for rate adaptation. The computation complexity at the source is $O(1)$. However, for n receivers, it needs $O(n)$ states at the source, and network aggregation can also lead to performance degradation. ERMCC does not have these drawbacks.

2.2 Multi-Rate Schemes

Ideally, the multi-rate multicast congestion control can satisfy heterogeneous receivers because each of them receives data at its own rate. The most noticeable among them are recently developed Fine-Grained Layered Multicast [6] and STAIR [7]. However, the multi-rate schemes are *closely coupled* with routing and IGMP, which implies some potential problems. For example, different groups for layers could follow different routes [13]. Aggregated multicast trees [14] do not necessarily prune trees dynamically and hence break the assumptions of the multi-rate schemes. The slackness of response to congestion due to long leave latency continues to be an issue. Besides, frequent group joins and leaves can introduce significant load at routers.

3 ERMCC

As we have mentioned in the introduction, in ERMCC, receivers send their average TRACs back to the sender whenever necessary, and the sender dynamically chooses a representative (CR) out of them and use only its TRACs to adjust the sending rate. In this section, we will present the details of how the whole scheme works. We will first present operations at an ERMCC receiver and at the source, followed by a list of the key features of ERMCC.

3.1 ERMCC Receiver

Receivers in ERMCC performs two major functions: (i) calculation and maintenance of TRAC and average TRAC, and (ii) proper generation and suppression of feedbacks to the source. The former function is crucial since TRAC is used to help the source in rate adaptation as well as in deciding which receiver will be the CR. The latter function is also important in that it determines scalability of ERMCC in terms of two well-known single-rate multicast problems: feedback-implosion, and slowest receiver tracking.

3.1.1 Throughput Rate At Congestion (TRAC) – $\mu(t)$, $\Omega(t)$, $\omega(t)$

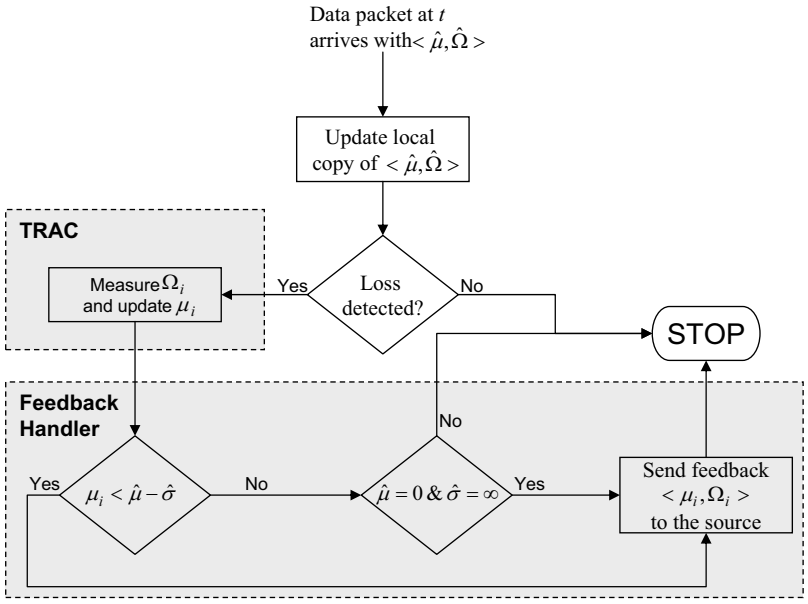
Upon detection of a packet loss at a receiver in ERMCC, that receiver measures explicit output rate TRAC $\Omega(t)$ and updates the average TRAC $\mu(t)$. We represent TRAC measured at time t at receiver i as $\Omega_i(t)$. Measurement of TRAC is done over a small time period Δt which we take as 1 second for all cases in this paper.

Thus, measurement of average TRAC $\mu(t)$ includes two levels of averaging. The first averaging is done to measure the TRAC, which can be expressed as averaging of instantaneous output rate $\omega(t)$. So, TRAC at receiver i at time t is calculated as:

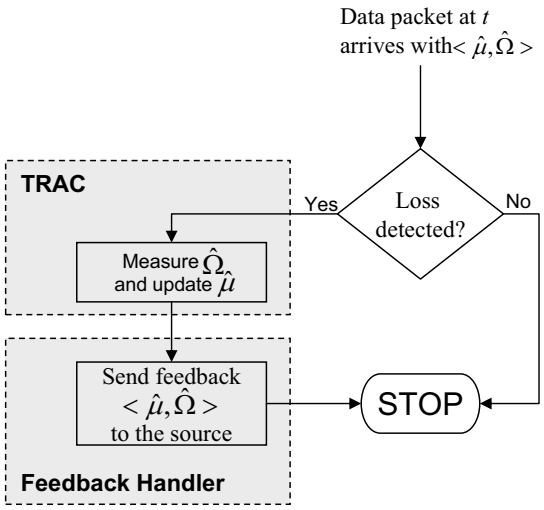
$$\Omega_i(t) = E[\omega_i(t)] = \frac{1}{\Delta t}\int_t^{t+\Delta t} \omega_i(t)dt \tag{1}$$

The second level of averaging is done by a moving average function $\Phi()$ (i.e. EWMA) with an exponential weighting factor of α. The value of α determines importance of the previous TRAC values in the resulting average. So, given that the previous packet loss happened at time t_0, average TRAC at receiver i at time t_1 is calculated by a recursive relationship:

$$\begin{aligned} \mu_i(t_1) &= \Phi(\mu_i(t_0), \Omega_i(t_1), \alpha) \\ \mu_i(t_1) &= (1-\alpha)\mu_i(t_0) + \alpha\Omega_i(t_1) \end{aligned} \tag{2}$$



(a) a non-CR receiver i



(b) the current CR receiver

Fig. 2. Operations at ERMCC receiver.

To distinguish these measures for CR we will use a hat on the notation for the rest of the paper. So, $\hat{\mu}(t)$, $\hat{\Omega}(t)$, and $\hat{\omega}(t)$ represents the average TRAC, the TRAC and the instantaneous output rate for the current CR of the multicast session.

Similar to average TRAC, another important metric to keep track of is the *deviation of TRAC*, because it plays a crucial role in feedback suppression as well as selection of CR which will be detailed in Sections 3.2.1 and 3.1.3 respectively. We represent the deviation of TRAC as $\sigma_i(t)$, and calculate it

again by means of the EWMA function $\Phi()$:

$$\begin{aligned}\sigma_i(t_1) &= \Phi(\sigma_i(t_0), \mu_i(t_1), \Omega_i(t_1), \alpha) \\ \sigma_i(t_1) &= (1-\alpha)\sigma_i(t_0) + \alpha|\mu_i(t_1) - \Omega_i(t_1)| \end{aligned} \tag{3}$$

3.1.2 Feedback Handler

In ERMCC, as shown in Figure 2-a, feedbacks are generated only when a packet loss is detected. Consider a data packet A at the arrival of which, receiver i detects that some data packets have been lost. The feedback generated by this receiver will contain: (i) the sequence number of the lastly received data packet A, (ii) the TRAC, $\Omega_i(t)$, measured at the arrival of A, and (iii) the average TRAC, $\mu_i(t)$. So, the feedback will be a tuple of three items. When the feedback arrives at the source, the first item will be used for making RTT estimation for CR, the second item will be used for adjusting the transmission rate, and the last item will be used in the decision-making process of CR selection.

Regarding the meaning of feedbacks in ERMCC, there are two different situations for two different purposes:

- **Required Feedback from the CR:** As shown in Figure 2-b, when the CR detects a packet loss, it needs to send congestion indication as a feedback to the source; so that the source can adjust the transmission rate. Since the feedback includes $TRAC\ \hat{\Omega}(t)$, it also serves as a congestion indication since it is measured up on detection of congestion.
- **Optional Feedback:** As shown in Figure 2-a, a non-CR receiver detects a packet loss and generates a feedback only when it thinks that it is slower than the current CR. For receiver i, the necessary condition for sending a feedback is $\mu_i(t) < \hat{\mu}(t) - \hat{\sigma}(t)$. Each non-CR receiver performs this comparison to make effective suppression of unnecessary feedbacks, which we will discuss next.

3.1.3 Feedback Suppression

Effective feedback suppression can reduce the risk of feedback implosion, and allow a multicast congestion control scheme to be used for large groups. In ERMCC, the source conveys the average TRAC $\hat{\mu}(t)$ and the deviation $\hat{\sigma}(t)$ of the CR's TRAC to receivers whenever the CR is updated or $\hat{\mu}(t)$ and $\hat{\sigma}(t)$ are changed. The source conveys these statistics about the current CR by attaching them to the data packets. A receiver will send feedbacks, only if its own average TRAC is less than the current average TRAC of the CR by an amount at least the standard deviation of the CR's TRAC. That is, for

receiver i the necessary condition for sending a feedback is $\mu_i(t) < \hat{\mu}(t) - \hat{\sigma}(t)$. Note that we do not use a weaker condition of $\mu_i(t) \leq \hat{\mu}(t)$ to be conservative and keep CR stable.

If needed, the source can use this behavior of the receivers to obtain feedbacks from all receivers. $\hat{\mu}(t)$ and $\hat{\sigma}(t)$ conveyed by the source can be changed to large or smaller values so that receivers can send feedbacks. This is needed when the current CR is inactive and the source needs to trigger feedbacks from all receivers for new CR selection (Figure 6). To remedy the possiblity of feedback implosion, the source can change these $\hat{\mu}(t)$ and $\hat{\sigma}(t)$ thresholds to obtain feedback from a portion of receivers at a time.

Clearly, no timer is involved in our feedback suppression, no knowledge of the whole group is needed. Unlike other probabilistic timer-based feedback suppression schemes, feedbacks are not scheduled at all before being suppressed. Yet, it is effective since the amount of feedbacks sent to the source is independent of the total number of receivers. More insight will be given in the theoretical analysis at Appendix B.5.

There is one situation which might be of concern. When the current CR is absent and the source needs to choose a new CR, all receivers seeing congestion of similar degree may send feedback at the same time. However, this situation will last at most one RTT, because the new CR will be chosen in one RTT. Besides, in reality, due to the heterogeneity of the network, many (if not most) receivers will get the information of the new CR before they can send out feedbacks for CR re-selection. Therefore, the total number of feedbacks sent under this situation is limited, and we do not deem it as a problem.

3.2 ERMCC Source

In a single-rate multicast congestion control protocol, the source is responsible for several major functions. These include: (i) proper and scalable selection of the CR that represents the slowest receiver(s) in the multicast session, (ii) proper adaptation of the transmission rate so that available bandwidth utilized as much as possible while assuring that the slowest receiver(s) is not overloaded, and (iii) estimation and maintenance of necessary statistics such as RTT. In order to perform the first function, ERMCC employs a set of *CR Selection* criteria as well as a *CR Mode Control* module that operates at every RTT. Similarly, to perform the second function, ERMCC has a *Rate Increase* module that operates at every RTT and a *Rate Decrease* module that operates at every congestion indication from the receivers.

As it is shown in Figure 3, an ERMCC source has six major functions and modules, each of which has a specific purpose. In the following subsections,

we will describe each of these functions and modules in detail.

3.2.1 CR Selection: Tracking of the Slowest Receiver

ERMCC compares average TRAC of all receivers to locate the slowest ones, and chooses one of them as the *Congestion Representative (CR)*. By using a metric like TRAC (which is based on explicit output rate), it avoids computing TCP throughput formula [11] [10] which requires per receiver RTT and packet loss rate.

ERMCC receivers help the source to select a receiver with the lowest average TRAC by sending in feedbacks *only if* their average TRACs are low enough to qualify them as CR. It is imperative that the receivers do not send more than necessary or less than enough feedbacks, which necessitates proper and effective suppression of feedbacks. Details of how receivers suppress the feedbacks was covered in Section 3.1.3.

Thus, to make selection of the slowest receiver as the CR, two types of comparisons take place in the system:

- *Comparison at receivers:* Each receiver checks whether it thinks itself as a potential CR. If so, it sends feedback to compete for being the CR.
- *Comparison at the source:* The source compares the feedbacks from those receivers who think they are qualified, and makes the final decision of which should be the CR.

These comparisons are shown in detail in Feedback Handler part of Figure 2-a and CR Selection part of Figure 5.

Network conditions always keep changing, and we need to continuously keep our choice of CR up-to-date. There are mainly two situations under which CR needs to be updated:

- **Case 1:** *A non-CR receiver worsens.* The situations of some non-CR receivers change so that one of them sees more severe congestion than the current CR does.
- **Case 2:** *CR improves or leaves.* While the situations of all non-CR receivers remain unchanged, the previously most congested path is improved so that the current CR sees less congestion than other receivers, or it leaves the multicast session.

Tracking the slowest receiver by examining average TRACs can deal with *Case 1*, but to cope with *Case 2* needs more effort. Under this situation, there can be no feedbacks from the current CR. Recall that the source only considers the feedbacks from the CR for rate adaptation and ignores all other feedbacks.

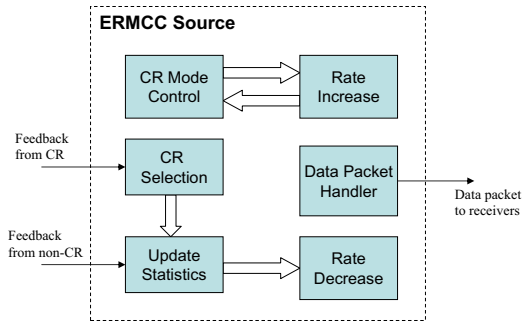


Fig. 3. Source operations as a block diagram.

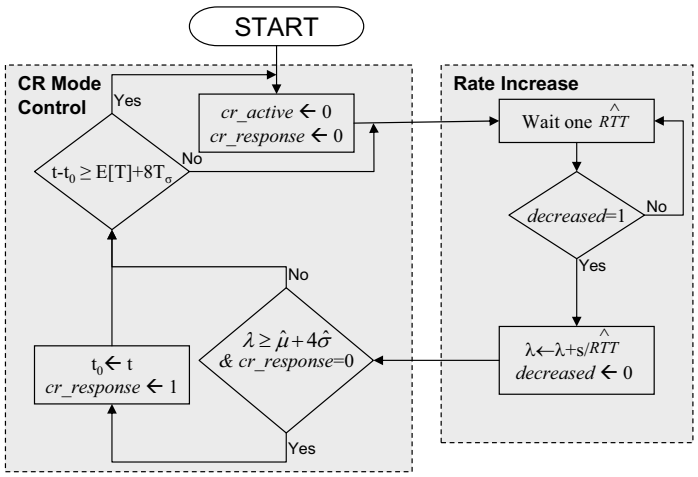


Fig. 4. At every RTT, the source attempts to increase the transmission rate and updates the operating mode of the source as either *CR Active* or *CR Inactive*.

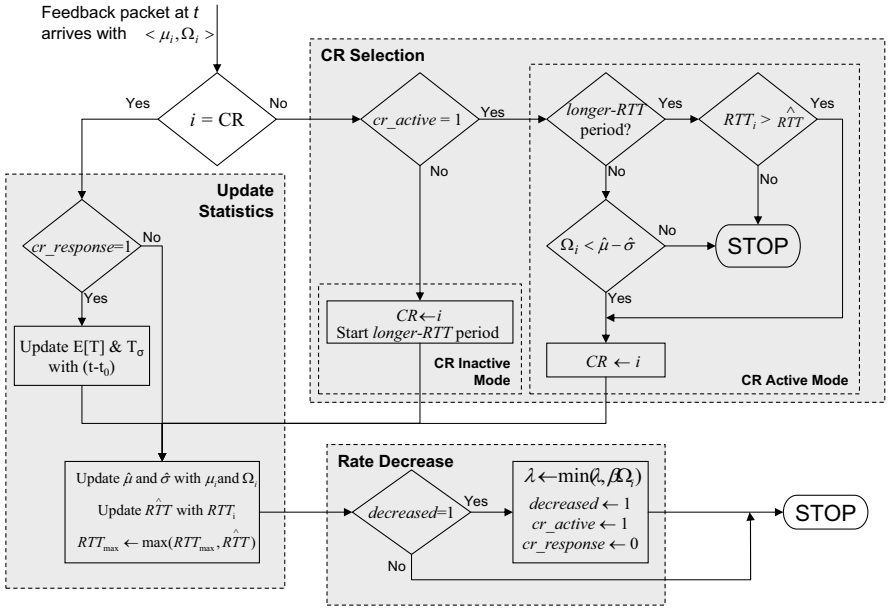


Fig. 5. Operations that take place when a feedback packet from receiver i arrives at the source.

If the source does not change CR in time, the transmission rate will be out of control. To detect this situation, we estimate an upper bound (denoted as T_{max}) of the idle time (denoted as T) before the source receives the first feedback from the CR when the bottleneck is fully loaded. Notice that T is indeed response time of CR during a congestion epoch, so we named it *CR Response Time*. We will give a detailed description of measurement of T later in Section 3.2.4.

As shown in CR Selection part of Figure 5, the source in ERMCC defines two modes for the CR, Active or Inactive, which reflect validity of the CR. At every RTT, the source updates the mode of CR. We will detail the update of CR's operation mode in the next Section 3.2.2.

There is one small trick we use to bias the choice of CR towards those receivers with higher RTTs. As shown in CR Inactive Mode part of the CR Selection in Figure 5, right after a new CR is chosen, we start a *longer-RTT* period of $2RTT_{max}$, where RTT_{max} is the maximum RTT the source has ever seen. Later within this period, as shown in CR Active Mode part of the CR Selection in Figure 5, if the source receives a feedback from another receiver with similar average TRAC as that of the CR, it will update CR to this receiver, since this one tends to have longer RTT. Notice that the longer-RTT period is not reset after CR switches within the longer-RTT period.

3.2.2 CR Mode Control

To determine whether or not the selected CR is active, the source uses two measures: (i) an estimate of the time when the bottleneck becomes fully loaed, and (ii) T_{max}, an estimate of the time it would maximally take the current CR to respond during congestion. Basically, the source starts to count when it detects the time corresponding to the first estimate above. And then, it identifies the CR as Inactive when the count reaches the second time estimate above. In other words, suppose we somehow detect that the bottleneck is fully loaded at time t. If there has been no feedback from the current CR until $t + T_{max}$, we can say that the current CR is now inactive and needs to be changed. Indeed, this is sort of a timeout on TRAC of the current CR. This process of mode determination can be seen in the flowchart shown in CR Mode Control part of Figure 4.

To see this mode control process on a timeline, let's look at Figure 6. When the CR is still active, we measure samples of T at the source, using feedback packets only from CR. When the transmission rate reaches $\hat{\mu}(t) + 4\hat{\sigma}(t)$ [3], we assume that bottleneck becomes fully loaded and start to count. Let the

[3] According to Chebychev's Inequality, about 94% of the random samples are less than this value.

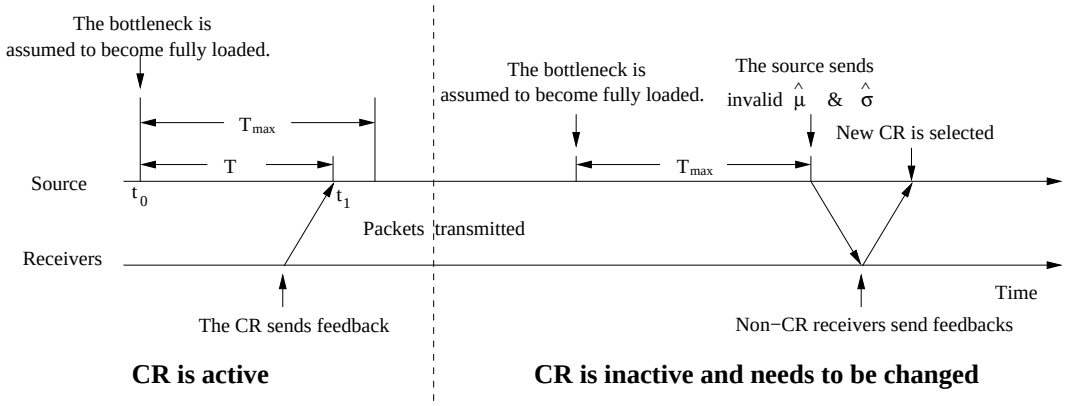


Fig. 6. Sketch of updating Congestion Representative (CR)

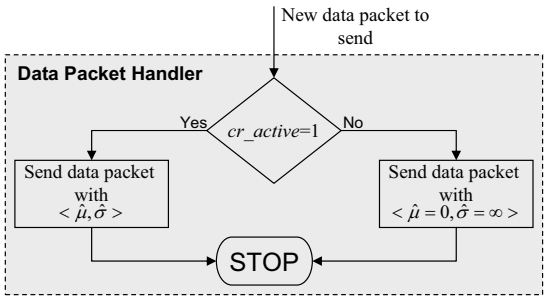


Fig. 7. Handling of data packets at the source: Source keeps attaching $\hat{\mu}$ and $\hat{\sigma}$ to every data packet.

current time be t_0. At a later time t_1, suppose the first feedback from the CR arrives at the source. Then, $t_1 - t_0$ is a sample of T and we update the average and deviation of T again with EWMA just like we did for the TRAC in (2) and (3). T_{max} is the average value of T plus eight times its deviation [4], i.e. $T_{max} = E[T] + 8T_\sigma$.

When the CR is not active, for the duration of T_{max} since we start to count, no feedback will be received by the source. The source then requests feedback from other receivers for new CR selection, as described in Section 3.1.3.

3.2.3 Rate Adaptation

Since TRACs are measured at receivers upon packet losses, they indicate how much bandwidth a flow can get out of the fully loaded bottleneck, assuming congestion is the only reason for packet losses. The less it can get, the more congested the bottleneck is. Therefore, we choose one receiver with the lowest average TRAC as the CR, and let the source only consider the feedbacks from that receiver for rate adaptation.

[4] We choose the value of 8 to be conservative.

ERMCC is a *rate-based* scheme, using the policy of additive increase and multiplicative decrease (AIMD). As shown in Rate Increase part of Figure 4, if there are no feedbacks from the CR, the transmission rate is increased by s/RTT per RTT, where s is the packet size, RTT is that between the source and the CR. If a feedback is received from the CR at time t_1, let the TRAC in this feedback be $\hat{\mu}(t)$, we adjust the transmission rate to the minimum of $\beta\hat{\mu}(t)$ and the current rate. Feedbacks from other non-CR receivers will be ignored, and at most one rate cut is allowed per RTT. This is shown in Rate Decrease part of Figure 5.

Thus, adaptation of the source rate $\lambda(t)$ is done according to the following AIMD-like method:

$$\lambda(t_1) = \begin{cases} \lambda(t_0) + s/RTT, & \text{no feedback} \\ min(\lambda(t_0), \beta\hat{\mu}(t_0)) & \text{feedback with } \hat{\mu}(t) \end{cases}$$

where the feedback $\hat{\mu}(t)$ arrives at source between t_0 and t_1, i.e. $t_0 < t \leq t_1$.

The rate reduction factor β is an important parameter of ERMCC. The larger the β, the more aggressive is ERMCC. To keep ERMCC TCP-friendly, from a later discussion in Appendix B.3, we will see that β must be at least 0.5. Moreover, the exact value of β depends on how ERMCC is implemented. According to the simulation and experiment results, we suggest $\beta = 0.65$ for implementation on user level, and $\beta = 0.75$ for implementation in system kernel. The reason is that, if ERMCC is implemented on user level, due to the coarseness of timers, its traffic is more bursty than that of TCP running in kernel. To cancel that effect, β should be set lower.

3.2.4 Update of Statistics

ERMCC source needs to maintain sets of statistics for the purposes of (i) estimating the RTT between the source and the CR, (ii) estimating response time of the CR during congestion epochs, and (iii) keeping track of the TRAC of the CR. Flowchart of how these statistics are updated is shown in Update Statistics part of Figure 5. We now briefly describe how each of these sets of statistics are updated:

RTT Estimation: Unlike a NAK, which includes the sequence number of a lost packet, a feedback in ERMCC includes the sequence number of a packet upon the arrival of which packet losses are detected. The source calculates the difference between the sending time of this packet and the arriving time of this feedback to get a sample of RTT. By doing this, we avoid the unnecessary delay between the supposed arriving time of a lost packet and the time of its loss being detected. Nevertheless, since feedbacks are sent only when packet losses

occur, RTT estimated by these feedbacks includes the maximum bottleneck queueing delay and thus is still the upper bound. On the other hand, ACKs as those in TCP may or may not include bottleneck queueing delay. Therefore, on average, RTT estimated by ERMCC's feedbacks is larger than that by ACKs under the same situation. In fact, this is the reason why we set β to some value higher than 0.5.

ERMCC source maintains the following two values regarding RTT: (i) $\widehat{RTT}$, estimate of the RTT between the source and the CR, and (ii) RTT_{max}, the maximum RTT estimate $\widehat{RTT}$ that was ever seen by the source. As shown in Figure 5, *upon receipt of a feedback* from receiver i, the source updates $\widehat{RTT}$ and RTT_{max} when either (i) the receiver i is the CR or (ii) the feedback caused the CR to be changed. Notice that this method calculates the RTT only from the samples when congestion exists.

CR Response Time: Another statistic that ERMCC source needs is the time, T_{max}, it would *maximally* take the current CR to respond during a congestion epoch. This is a crucial measure since it is used to determine whether or not the current CR is still active or not, as it can so happen that the CR may leave the system. The value of T_{max} is composed of $E[T]$ and T_σ which are average value of T and its deviation respectively. The composition we use is $T_{max} = E[T] + 8T_\sigma$, which means the source needs to measure and maintain the values of $E[T]$ and T_σ. As it can be seen from Update Statistics part of Figure 5, the source updates $E[T]$ and T_σ only *upon receipt of a feedback* from the current CR within the time period that started when the bottleneck is estimated to be fully loaded after a rate increase.

CR's TRAC: As described in (2), average TRAC is calculated by means of an *EWMA* function, which we represent as $\Phi()$. In addition to average TRAC, the source also maintains the deviation of TRAC, $\hat{\sigma}$, for the current CR. CR's average TRAC, $\hat{\mu}$, and deviation of CR's TRAC, $\hat{\sigma}$, are crucial statistics since they represents the maximum possible transmission rate for the current session and are directly used for the process of CR selection. As shown in Figure 5, *upon receipt of a feedback* from receiver i, the source updates $\hat{\mu}$ and $\hat{\sigma}$ when either (i) the receiver i is the CR or (ii) the feedback caused the CR to be changed.

3.2.5 Data Packet Handler

Receivers in ERMCC must be informed about the current value of CR's TRAC, $\hat{\mu}$, and its deviation, $\hat{\sigma}$. In order to convey $\hat{\mu}$ and $\hat{\sigma}$ to the receivers, ERMCC source attachs them to the data packets. As shown in Figure 7, the source specifically sets $\hat{\mu} = 0$ and $\hat{\sigma} = \infty$ when the CR is Inactive mode. The purpose of this is to make the receivers send their current TRAC values, so

that a new CR can be elected.

Even though we have not implemented in the simulations of this paper, it is also possible to set $\hat{\mu}$ and $\hat{\sigma}$, so that only those receivers with TRAC very close to the latest CR's TRAC will send feedback. Such a strategy is particularly needed when the total number of receivers is too large.

3.3 Key Features of ERMCC

As we can see from the details above, ERMCC has the following features:

- $O(1)$ **Memory Complexity:** The amount of memory needed to maintain the state information at source and receivers is $O(1)$. That is, the number of states is constant and independent of the number of receivers in a multicast session.
- **Practical Operations:** Operations of source and receivers are all simple, without requiring intense computation. In particular, there is no need to do per-receiver RTT estimation.
- **Effective Feedback Suppression:** With our non-probabilistic-timer-based feedback suppression mechanism in place, the amount of feedbacks is independent of the total number of receivers.

The pseudo code of ERMCC's algorithm is provided in Appendix A for reference. The code for *ns*-2 and Unix can also be found at [15].

4 Properties about ERMCC Performance

It is desirable to check the performance of a multicast congestion control scheme by theoretical analysis. We have done that for ERMCC to show the following properties:

Property 1 *ERMCC is capable of tracking the slowest receiver and select it as* CR (Congestion Representative) *to direct rate adaptation.*

PROOF. See Appendix B.2.

Property 2 *ERMCC is TCP-friendly on the representative path, i.e. the path between the source and the CR.*

PROOF. See Appendix B.3.

Property 3 *ERMCC is immune to drop-to-zero problem, i.e. the sending rate will not be reduced more than enough and converge toward zero upon asynchronous congestion.*

PROOF. See Appendix B.4.

Property 4 *Feedback suppression in ERMCC is very effective.*

PROOF. See Appendix B.5.

5 Simulations and Experiments

We have run simulations on *ns*-2 [16] and experiments in Emulab [9] to validate the performance of ERMCC. The *ns*-2 simulations checked the following aspects:

(1) TCP-Friendliness
(2) Drop-to-zero avoidance
(3) Multiple bottleneck fairness
(4) Slowest receiver tracking
(5) Feedback suppression

We also ran the same set of *ns*-2 simulations for PGMCC[2] and TFMCC[3] and compared the performance of our scheme with theirs. For ERMCC and TFMCC, we use ns2.1b7a, for PGMCC, we use ns2.1b5, due to the restriction of its source code. In all simulations, the data packet size is 1000 bytes, the bottleneck buffer size is 50K bytes, the initial RTT is 100 milliseconds.

For experiments on real systems in Emulab[9], we implemented ERMCC on top of UDP as a user level program. TCP-friendliness and drop-to-zero behavior are tested. The result is presented at the end of this section.

5.1 TCP-Friendliness and Drop-To-Zero Avoidance

We used a star topology (Figure 8) to generate asynchronous and independent congestion on different paths. There are 129 ends nodes in the topology. Between each pair of source i and receiver i ($i = 1 \ldots 64$), there are one TCP Reno flow and one single-receiver ERMCC flow. Furthermore, there is a multi-receiver ERMCC flow from source 65 to all 64 receivers. Therefore, on a path

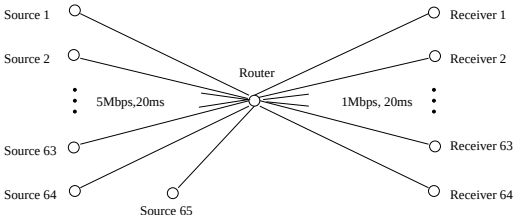


Fig. 8. 64-receiver star topology with TCP background traffic.

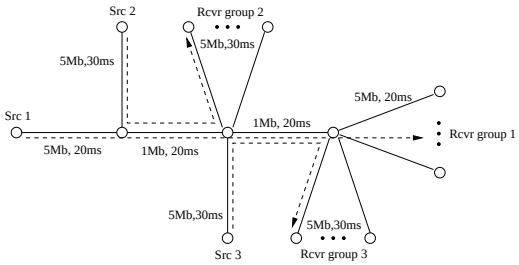


Fig. 9. Linear network with multiple bottlenecks including a total of 48 receivers.

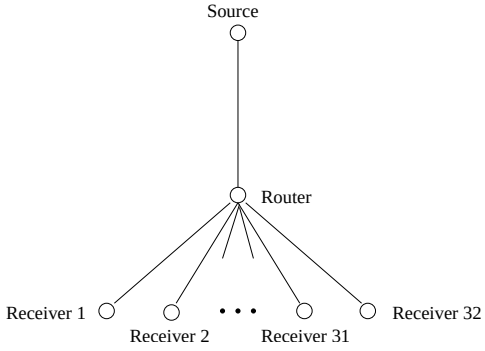


Fig. 10. One-level tree with 32 receiver nodes.

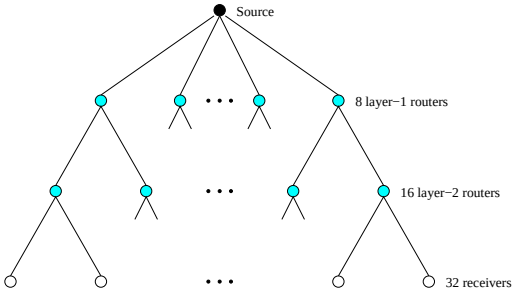


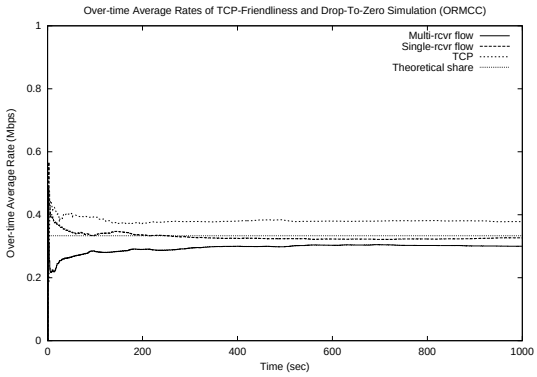
Fig. 11. Heterogeneous dynamic network.

between the router and any receiver, the multi-receiver ERMCC flow competes with a TCP flow and a single-receiver ERMCC flow.

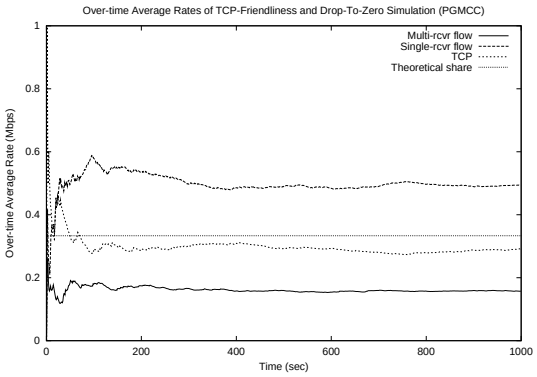
We randomly chose a receiver node and plot in Figure 12(a) the over-time average rates [5] of all three flows going to it. The fact that the average rates of the TCP flow, the single-receiver ERMCC flow and the multi-receiver ERMCC flow are close to each other indicates that *(1) ERMCC is TCP-friendly, and (2) ERMCC does not suffer from drop-to-zero problem.*

We also conducted experiments on the same configuration for PGMCC and TFMCC. For PGMCC simulations, since ns2.1b5 can only accommodate up to 128 end nodes, we can only have 63 pairs of unicast source and receiver instead of 64 pairs. Moreover, we only measure the sending rate of original data packets, because repair packets for PGMCC are routed by net elements to individual receivers whoever need them instead of all receivers. Nevertheless, the proportion of repair packets is less than 1/10 and is thus negligible. This has the same effect as measuring sequence number increment in [2]. Results in Figure 12 (b) and (c) show that the average rates of PGMCC and TFMCC multicast flows deviate more from the corresponding unicast flows than what ERMCC does.

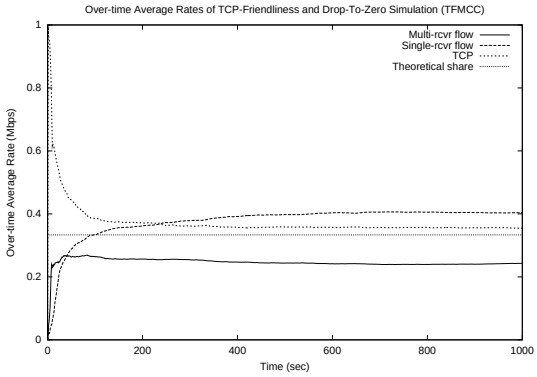
[5] Over-time average rate at time t is defined as the traffic volume between $[0, t]$ divided by t.



(a) ERMCC



(b) PGMCC



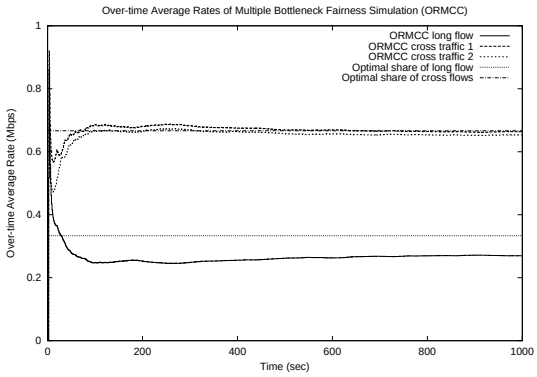
(c) TFMCC

Fig. 12. TCP-friendliness and immunity to drop-to-zero: ERMCC is more TCP-friendly and avoids drop-to-zero better than PGMCC and TFMCC.

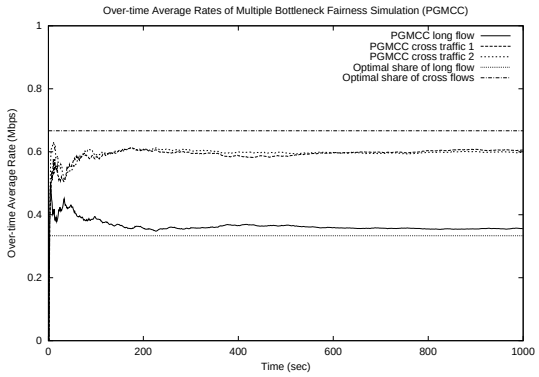
5.2 Multiple Bottleneck Fairness

In real world, there are usually more than one bottleneck on a path. It is desirable to check how long ERMCC flows compete with short ones and what kind of fairness ERMCC can achieve. We ran a simulation on the topology shown in Figure 9. There is a long multi-receiver multicast flow, going through two bottlenecks, from Src 1 to receivers in Group 1. There are also two short multicast flows going through only one bottleneck from Src 2 to Group 2 and from Src 3 to Group 3 respectively. Each group has 16 receivers. RED queues are used on the routers to reduce the effect of RTT estimation.

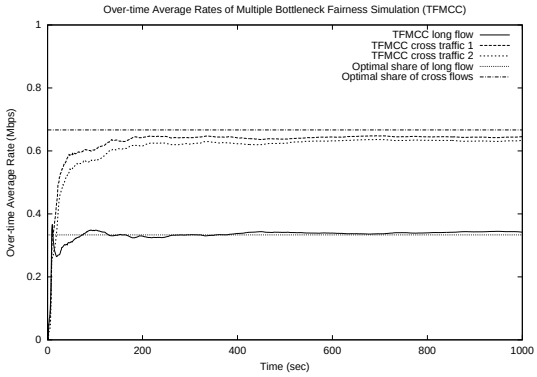
According to proportional fairness, the long ERMCC flow should get one-third of the bottleneck bandwidth, 0.33Mbps. The result in Figure 13 (a) shows that ERMCC achieves approximately proportional fairness. Similar fairness achieved by PGMCC and TFMCC in the same configuration is also shown in the figure.



(a) ERMCC



(b) PGMCC



(c) TFMCC

Fig. 13. Fairness of sharing bottleneck bandwidth: All three schemes achieve approximately proportional fairness.

5.3 Slowest Receiver Tracking

This simulation is used to test ERMCC's capability to quickly track the slowest receiver and select it as CR. In the tree topology shwon in Figure 10, there is an ERMCC flow between the source and all the 32 receivers. There are three dynamically generated bottlenecks using TCP Reno flows. Denote link i as the link between the router and receiver i ($i = 1 \ldots 32$), each link has 2Mb bandwidth and 20 ms delay. The simulation time is 1000 seconds. During the whole simulation, one TCP flow runs on link 1; between 200th and 800th seconds, three TCP flows run on link 2; between 400th and 600th seconds, seven TCP flows run on link 3.

The dynamics include both conditions causing CR switches, i.e. (1) A slower receiver appears, (2) The current slowest receiver is absent. RED and drop-tail queue management policies are used separately in our simulations. Simulation results are shown in Figure 14 (a) and (d). Vertical dashed lines show when

the ERMCC source switched CR. We can see that ERMCC updates CR and adapts its transmission rate in a timely manner. Note that when using RED queues, the ERMCC source sometimes switched CR a little slower than drop-tail situation. The reason is because RED queue drops packets in a random manner, it takes longer for the slowest receiver to have a lower average TRAC measurement.

Under the same situation, as shown in Figure 14 (b),(c), (e) and (f), PGMCC and TFMCC also track the slowest receiver, though sometimes with more representative switches. We also noticed that there is much oscillation of PGMCC's rates due to its design of mimicking TCP, while the rates of ERMCC and TFMCC have similar smoothness.

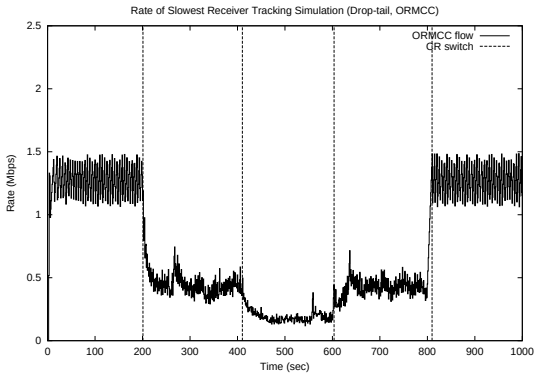
5.4 Feedback Suppression

To check the effectiveness of the feedback suppression mechanism in ERMCC, we refer back to the simulation of TCP-friendliness and drop-to-zero avoidance. In totally ten simulations, the average total number of feedacks sent by all receivers is 816 (standard deviation is 14.8), the average total number of suppressed feedbacks is 34601 (standard deviation is 422.0). The average number of feedbacks would have been sent by a receiver if without suppression, is $(34601 + 816)/64 \approx 553$. As we discussed in the analysis (Appendix B.5), realistic measurement error can lead to a little bit more feedbacks. Since $816 < 2 \times 553$, we can still say that the overall feedback volume with suppression is approximately equal to that from a single receiver if without suppression. The high ratio of feedbacks suppressed, $34601/(34601 + 816) \times 100\% \approx \mathbf{97.7}\%$, shows that *our feedback suppression is very effective.*

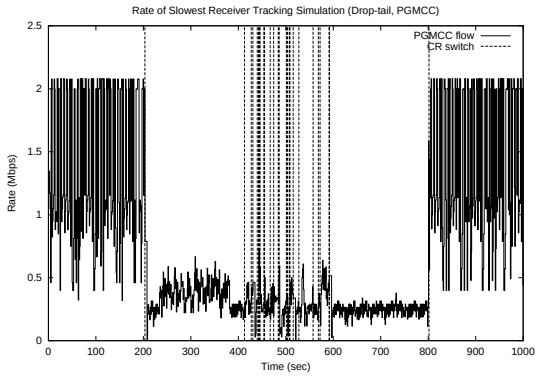
For comparison, in a typical PGMCC simulation with the same configuration, the total number of feedback packets received by the source is 74830 (NAK 55222, ACK 19608); for TFMCC, it is 5344. Their feedback volume is much larger.

5.5 Comparison with PGMCC and TFMCC in Heterogeneous Dynamic Network

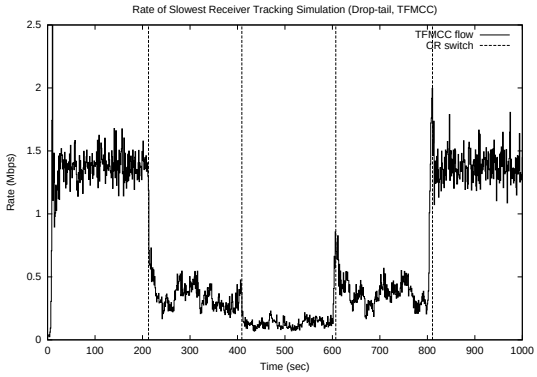
As the last simulation, we constructed a dynamic network to test the stability and adaptability of ERMCC, again compared with PGMCC and TFMCC. In Figure 11, each link has 2Mbps bandwidth. 2 links at the first level, 4 links at the second level, and 8 links at the third level has 200ms delay. All other links have 20ms delay, while on any path between the source and a receiver, there is at most one link of 200ms delay. On each link, two TCP Reno flows



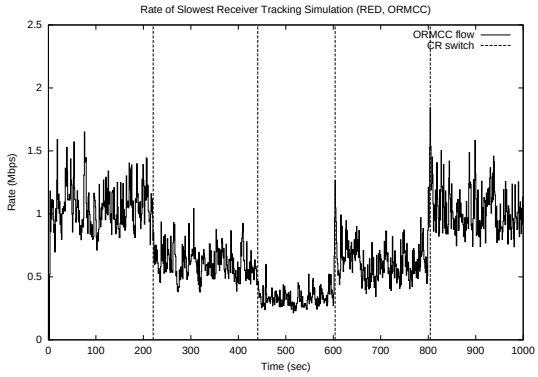
(a) Droptail Queue (ERMCC)



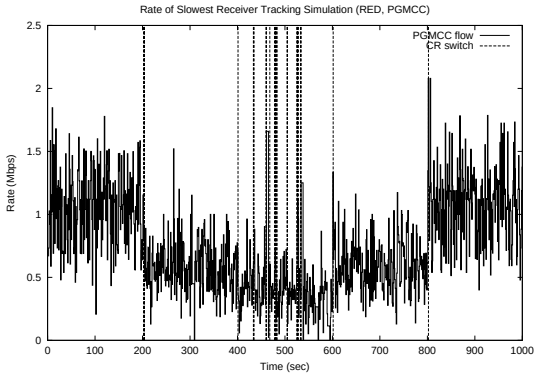
(b) Droptail Queue (PGMCC)



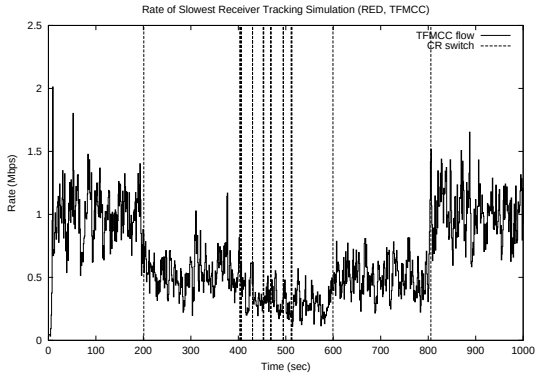
(c) Droptail Queue (TFMCC)



(d) RED Queue (ERMCC)



(e) RED Queue (PGMCC)



(f) RED Queue (TFMCC)

Fig. 14. Capability of tracking the slowest receiver: ERMCC tracks the slowest receiver in time with fewer representative switches. The most congested bottleneck is changed at every 200s. For the five 200s periods of the 1000s simulation, the most congested bottlenecks are link 1, link 2, link 3, link 2, and link 1 in the topology shown in Figure 10.

are randomly turned on and off according to Pareto distribution with average value of 60 seconds, and two UDP flows of 200Kbps on and off with average

Table 2
COMPARISON OF AVERAGE THROUGHPUT AND FEEDBACK VOLUME IN HETEROGENEOUS DYNAMIC NETWORK (ERMCC has higher throughput and less feedback.)

	ERMCC	PGMCC	TFMCC
Average Throughput	**415.4** Kbps	126.6Kbps	226.7Kbps
Average Number of Feedbacks	**866.9**	5009.6 (NAK only)	3312.9

value of 1 second. These flows dynamically generate bottlenecks and make the network heterogeneous. At last, there is a multicast flow from the source to all the receivers. The multicast flow can use either ERMCC, PGMCC or TFMCC. Therefore, at any moment, there are at most five flows on any link: one multicast flow, two TCP flows and two UDP flows, and the multicast flow is expected to get an average throughput rate of 500Kbps or so.

We ran 10 simulations for each of the three schemes. In Table 2 we can see that with smaller amount of feedbacks, ERMCC can achieve higher throughput. That means, *ERMCC has better stability and adaptability in heterogeneous and dynamically changing networks.*

5.6 ERMCC Implementation in Emulab

To do a preliminary check of ERMCC's performance in real world and understand the issues of implementation, we implemented ERMCC in C++ on top of UDP as a user level program and ran it in Emulab [9] [6]. The operating system we used is RedHat 7.1, and mrouted[17] is used for multicast routing. On the topology shown in Figure 15, the links between the peripheral nodes and their parent nodes have 50 ms propagation delay and 1.0Mbps bandwidth. All other links have 100Mbps bandwidth and 0 ms propagation delay. [7] From the center node to any peripheral node, there is a single-receiver ERMCC flow and a TCP flow. Also, there is a multi-receiver ERMCC flow from the center node to all 36 peripheral nodes. The experiment time is 1000 seconds.

Since we implemented ERMCC on user level, its traffic is more bursty than that of TCP running in kernel. As described in Section 3.2.3, the rate cut factor (β) should be adjusted accordingly. We tried three different values: 0.5, 0.65 and 0.75. According to Figure 16 [8], when $\beta = 0.5$, the TCP flows got more

[6] Emulab is accessible at http://www.emulab.net.
[7] The propagation delay here means the delay artificially introduced by some particular software.
[8] The mean and confidence interval are calculated out of all the flows of the same category.

bandwidth; when $\beta = 0.75$, the ERMCC flows are more aggressive. $\beta = 0.65$ works the best, where the multi-receiver ERMCC flow got almost the same bandwidth as TCP flows did, and thus TCP-friendly. Moreover, among all the values tested for β, the average rates of the multi-receivers ERMCC flow and single-receiver ones are always close, showing that ERMCC is immune to drop-to-zero problem.

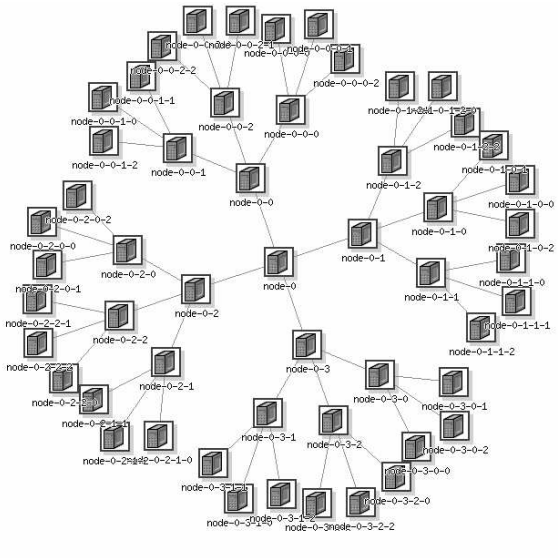
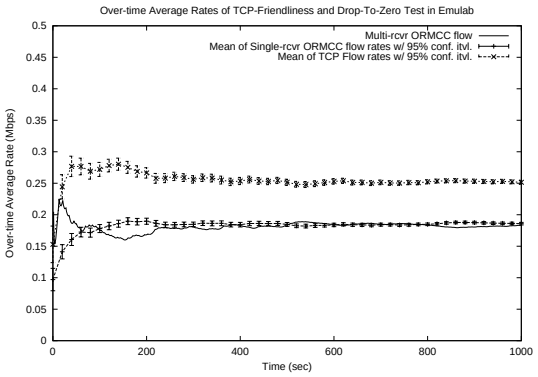


Fig. 15. Topology used in Emulab for TCP-friendliness and drop-to-zero tests with 36 receiver nodes.

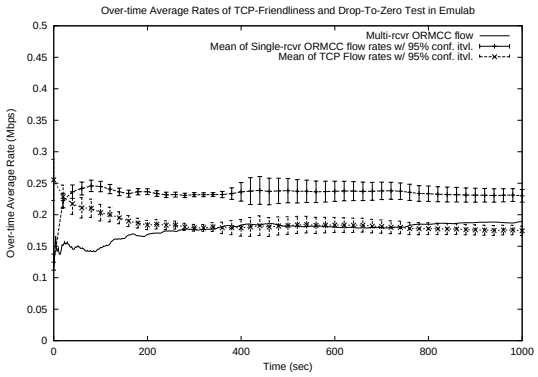
6 Conclusion

In this paper, we have proposed a single-rate multicast congestion control scheme, which uses a conventional concept of representative named *Congestion Representative (CR)*. However, by leveraging a new metric *TRAC*, the ERMCC scheme is capable of effectively addressing the problems of TCP-friendliness, drop-to-zero, slowest receiver tracking and feedback suppression. The states maintained by source and receivers are $O(1)$; operations of source and receivers are all simple without requiring intense computation. In particular there is no need to measure RTTs between all receivers and the source. ERMCC also shows that non-probabilistic-timer-based feedback suppression is highly effective. To confirm the performance of ERMCC, we have not only provided theoretical analysis, but also performed simulations to compare it with PGMCC[2] and TFMCC[3]. Furthermore, we have implemented ERMCC over UDP and ran it on real systems in Emulab[9]. Both simulation and implementation results show ERMCC's excellent performance. As an emphasis, we summarize the comparison with PGMCC and TFMCC in simulations in Table 3. We can see that ERMCC achieves better performance than PGMCC



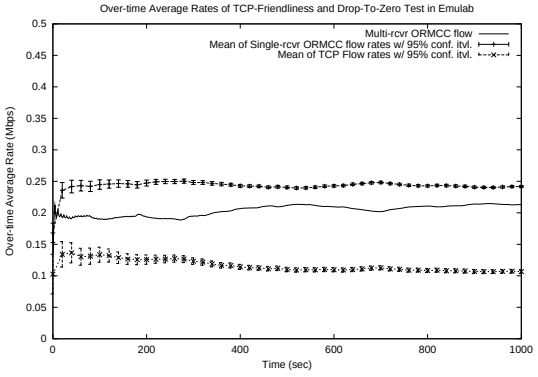
(a) $\beta = 0.5$

ERMCC is less aggressive than TCP



(b) $\beta = 0.65$

ERMCC is TCP-friendly



(c) $\beta = 0.75$

ERMCC is more aggressive than TCP

Fig. 16. TCP-friendliness and drop-to-zero test result in Emulab: ERMCC is TCP-friendly and avoids drop-to-zero on real systems with proper β setting.

and TFMCC under most situations. Code for *ns*-2 and Unix is available at [15] for public test.

We believe that further studies of ERMCC-like schemes will benefit the area of multicast congestion control. A point that deserve further investigation is the EWMA smoothing technique used at various places of the scheme. Particularly, it is worthwhile to study averaging techniques that can use the timestamp differences of arriving data packets at the receiver. Also, adaptive tuning of various parameters (e.g. less than 4σ in determining slowest receiver with its average TRAC) can provide incremental improvements to ERMCC.

Table 3
SUMMARY OF COMPARATIVE SIMULATIONS

Property	*Comparison w/ PGMCC & TFMCC*	*Figure/Table*
TCP Friendliness	+	Fig. 12
Immunity to Drop-to-Zero	+	Fig. 12
Multiple Bottleneck Fairness	+/-	Fig. 13
Slowest Receiver Tracking	+	Fig. 14
Feedback Suppression	+	Table 2
Heterogeneous Dynamic Network	+	Table 2

7 Acknowledgment

We would like to thank Joerg Widmer, Gianluca Iannaccone, Chin-ying Wang, Sonia Fahmy for providing the NS code for PGMCC and TFMCC and/or helping the setup. We also appreciate the help from operators in Emulab testbed, as well as the comment by our lab-mates Kartikeya Chandrayana, Satish Raghunath, Biplab Sikdar, Yong Xia and Tao Ye.

References

[1] D. DeLucia, K. Obraczka, A multicast congestion control mechanism using representatives, in: Proceedings of IEEE ISCC, 1998.

[2] L. Rizzo, Pgmcc: A tcp-friendly single-rate multicast congestion control scheme, in: Proceedings of ACM SIGCOMM, 2000.

[3] J. Widmer, M. Handley, Extending equation-based congestion control to multicast applications, in: Proceedings of ACM SIGCOMM, 2001.

[4] J. Macker, R. Adamson, A tcp friendly, rate-based mechanism for nack-oriented reliable multicast congestion control, in: Proceedings of IEEE GLOBECOM, 2001.

[5] P. Thapliyal, Sidhartha, J. Li, S. Kalyanaraman, Le-sbcc: Loss-event oriented source-based multicast congestion control, Multimedia Tools and Applications 17 (2-3) (2002) 257–294.

[6] J. Byers, M. Luby, M. Mitzenmacher, Fine-grained layered multicast, in: Proceedings of IEEE INFOCOM, 2001.

[7] J. Byers, G. Kwon, Stair: Practical aimd multirate multicast congestion control, in: Proceedings of NGC, 2001.

[8] J. C. Lin, S. Paul, Rmtp: A reliable multicast transport protocol, in: Proceedings of IEEE INFOCOM, 1996.

[9] B. W. et al, An integrated experimental environment for distributed systems and networks, in: Proceedings of USENIX OSDI, 2002.

[10] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling tcp throughput: A simple model and its empirical validation, in: Proceedings of ACM SIGCOMM, 1998.

[11] M. Mathis, J. Semke, J. Mahdavi, T. Ott, The macroscopic behavior of the tcp congestion avoidance algorithm, ACM Computer Communications Review 27 (3).

[12] T. T. Fuhrmann, J. Widmer, On the scaling of feedback algorithms for very large multicast groups, Computer Communications 24 (5) (2001) 539–547.

[13] T. Nguyen, K. Nakauchi, M. Kawada, H. Morikawa, T. Aoyama, Rendezvous points based layered multicast, IEICE Transactions on Communication E84-B (12).

[14] A. Fei, J. Cui, M. Gerla, M. Faloutsos, Aggregated multicast: an approach to reduce multicast state, in: Proceedings of IEEE GLOBECOM, 2001.

[15] Ormcc source code, http://networks.ecse.rpi.edu/source_code.html.

[16] S. B. et al, Improving simulation for network research, Tech. Rep. 99-702b, University of Southern California (September 1999).

[17] mrouted 3.9 beta3-1 and mrouted linux patch, ftp://ftp.rge.com//pub/communications/ipmulti/beta-test/mrouted-3.9-beta3.tar.gz, ftp://ftp.debian.org/debian/dists/potato/non-free/source/net/ mrouted_3.9-beta3-1.diff.gz.

[18] S. Floyd, V. Jacobson, C.-G. Liu, L. Z. S. McCanne, A reliable multicast framework for light-wight sessions and application level framing, IEEE/ACM Transactions on Networking 5 (6) (1997) 784–803.

[19] P. Sharma, D. Estrin, S. Floyd, V. Jacobson, Scalable timers for soft state protocols, in: Proceedings of IEEE INFOCOM, 1997.

[20] J. Nonnenmacher, E. W. Biersack, Scalable feedback for large groups, IEEE/ACM Transactions on Networking 7 (3) (1999) 375–386.

A Algorithm

A.1 Operations at Source

Some of the following operations take place when either a feedback packet from a receiver r is received, or an RTT time period has been completed:

Variables :

r : The receiver sending the received feedback

λ : Current transmission rate at the source

Ω_r : Throughput rate at congestion (TRAC) in the received feedback from r

$\hat{\mu}$: Average TRAC of the CR

$\hat{\sigma}$: Deviation of TRAC of the CR

s : Packet size

RTT_{max} : Maximum RTT

$\widehat{RTT}$: RTT between the source and the CR

T : CR response time when the bottleneck is fully loaded

$E[T]$: Average of T

T_σ : Deviation of T

cr_valid : Indicates whether the CR is valid

cr_response_timer : Indicates whether the bottleneck is estimated to be full

t_0 : The estimated time bottleneck started to fill up

t : Current time

Initialization:
cr_valid = **false**
$RTT_{max} = 0$
$t_0 = 0$
cr_response_timer = **false**

Event every $\widehat{RTT}$**:**
if *There is no rate reduction within the recent* $\widehat{RTT}$ **then**
$\lambda \leftarrow \lambda + s/\widehat{RTT}$
if $\lambda \geq \hat{\mu} + 4\hat{\sigma}$ **and** *cr_response_timer is* **false then**
$t_0 \leftarrow t$
cr_response_timer $\leftarrow$ **true**
endif
endif
if $t - t_0 \geq E[T] + 8T_\sigma$ **then**
cr_valid $\leftarrow$ **false**
cr_response_timer $\leftarrow$ **false**
endif

Send packet:
if *cr_valid is* **true then**
Send a packet with real $\hat{\mu}$ and $\hat{\sigma}$
else
Send a packet with invalid values for $\hat{\mu}$ and $\hat{\sigma}$
endif

Subroutine : CutRate ()
if λ *has not been cut within the most recent* $\widehat{RTT}$ **then**
$\lambda \leftarrow \min(\lambda, 0.75\Omega_r)$
cr_valid $\leftarrow$ **true**
cr_response_timer $\leftarrow$ **false**
endif

Subroutine : UpdateStats ()
Update $\hat{\mu}$ and $\hat{\sigma}$ with Ω_r
Update $\widehat{RTT}$ with RTT_r
if $RTT_{max} < \widehat{RTT}$ **then**

$RTT_{max} \leftarrow \widehat{RTT}$
endif

Event upon receipt of feedback from r**:**
if r *is CR* **then**
if *cr_response_timer is* **true then**
Update $E[T]$ and T_σ with $(t - t_0)$
endif
do UpdateStats ()
do CutRate ()
return
endif

/* The feedback is NOT from CR */
if *cr_valid is* **false then**
Choose r as the CR
Start CR grace period as $2RTT_{max}$
else if *In CR grace period* **then**
if $RTT_r > \widehat{RTT}$ **then**
Choose r as the CR
endif
/* NOT in longer RTT period */
else if $\Omega_r < \hat{\mu} - \hat{\sigma}$ **then**
Choose r as the CR
endif

if *CR has been changed at the receipt of this feedback* **then**
do UpdateStats ()
do CutRate ()
endif

A.2 Operations at Receiver i

The following operations take place when a data packet is received at receiver i:

Variables :

Ω_i : A throughput rate at congestion (TRAC) sample

μ_i : Average TRAC of this receiver

$\hat{\mu}$: Average TRAC of the CR

$\hat{\sigma}$: Deviation of TRAC at the CR

Event upon receipt of a packet:
if $\hat{\mu}$ *and* $\hat{\sigma}$ *has been changed* **then**
Update the local copy of $\hat{\mu}$ and $\hat{\sigma}$
endif
if *This packet indicates packet losses* **then**
Measure Ω_i and update μ_i
if $\hat{\mu}$ *and* $\hat{\sigma}$ *are invalid* **or** $\mu_i < \hat{\mu} - \hat{\sigma}$ **then**
Send a feedback to the source
endif
endif

Table B.1
ADDITIONAL MATHEMATICAL NOTATIONS

Symbol	*Meaning*
R_i	The receiver behind path i
W_i	Bandwidth of the bottleneck on path i
Q_i	Buffer size of the bottleneck on path i
$\widehat{RTT}$	RTT on the representative path
RTT_i	RTT on path i
$\lambda_i(t)$	ERMCC flow's sending rate at the bottleneck on path i at time t
Δ	ERMCC flow's sending rate increment per $\widehat{RTT}$
$\lambda_i^o(t)$	Sum of the sending rates of all flows except the ERMCC flow sharing the bottleneck on path i at time t
Δ_i^o	The sum of the sending rate increments/decrements per unit time of all but the ERMCC flows sharing the bottleneck on path i
γ_i	$\gamma_i = \Delta_i^o/\Delta$
M	Total number of receivers behind MCB
RTT_i^f	Forward latency towards the receiver R_i
$\bar{\lambda}$	ERMCC flow's average sending rate
p	Packet loss rate experienced by the receivers behind MCB

B Theoretical Analysis

B.1 Expected TRAC

To build up for later analysis, in this part, we analyze the expected sending rate of an ERMCC flow.

Suppose there are $N > 1$ different paths from the source on the multicast tree, and also assume that there is a single receiver per path. Let R_i be the receiver behind path i. For convenience, we are going to refer the path between the source and the CR as *Representative Path*. Without loss of generality, assume R_1 is the current CR, and hence path 1 is the Representative Path. The source will choose another receiver R_j ($j \neq 1$) as the new CR only if R_j sees a lower average TRAC than that seen by R_1.

Consider the major bottleneck link of path i, which has a bandwidth of W_i with

a buffer size of Q_i.[9] Let $\lambda_i(t)$ be the ERMCC flow's sending rate observed at the bottleneck on path i at time t.[10] Similarly, let $\lambda_i^o(t)$ be sum of the sending rates of all flows except the ERMCC flow sharing the bottleneck on path i at time t. Also, let Δ be ERMCC flow's positive sending rate increment per unit time, i.e. $\Delta = s/\widehat{RTT} > 0$. Similarly, let Δ_i^o be the sum of the sending rate increments/decrements per unit time of all but the ERMCC flows sharing the bottleneck on path i. Depending on the other flows' behavior, Δ_i^o will change randomly.

To simplify the analysis, we assume that data are sent bit-by-bit evenly, sending rates are increased continuously, as well as that all packet losses are due to congestion. Also, drop-tail buffer management is assumed for bottlenecks.[11]

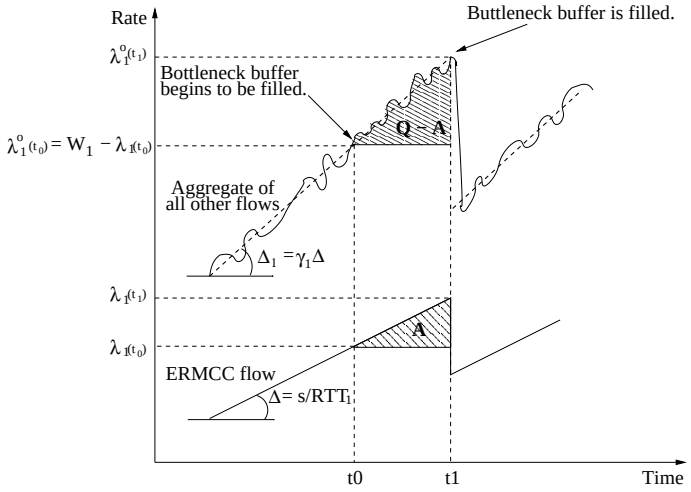


Fig. B.1. Evolution of the ERMCC flow's sending rate on the representative path.

Let's consider path 1 first (Figure B.1). Suppose at time t_1, there is a burst of packet losses, which means the bottleneck queue must be full at this moment. This also implies that the sum of sending rates of all the flows going through the bottleneck, must be larger than the bottleneck bandwidth, i.e. $\lambda_1(t_1) + \lambda_1^o(t_1) > W_1$. It follows that at an earlier moment t_0, the sending rate sum must have been equal to the bottleneck capacity W_1, i.e.,

$$\lambda_1(t_0) + \lambda_1^o(t_0) = W_1 \tag{B.1}$$

[9] We assume that all bottlenecks in the multicast session have the same buffer size. If a queue is constantly non-zero, we will treat the part which is emptied and (partly) filled as the whole queue.

[10] Note that this is different than the source's sending rate due to the possible bottleneck location and propagation delays from the source.

[11] Although our analysis is based on drop-tail routers, ERMCC also works well with RED routers. It has been confirmed by simulations, though for space reasons, the results are not included.

Since the sending rate of the ERMCC flow grows by Δ per unit time,

$$\lambda_1(t_1) = \lambda_1(t_0) + (t_1 - t_0)\Delta \Rightarrow \lambda_1(t_0) = \lambda_1(t_1) - (t_1 - t_0)\Delta \tag{B.2}$$

On average, $\lambda_1^o(t)$ grows by $\Delta_1^o = \gamma_1\Delta$ per unit time. Therefore,

$$\lambda_1^o(t_1) = \lambda_1^o(t_0) + (t_1 - t_0)\Delta_1^o = \lambda_1^o(t_0) + (t_1 - t_0)\gamma_1\Delta \tag{B.3}$$

From (B.1), (B.2) and (B.3), we have,

$$\begin{aligned}\lambda_1^o(t_1) &= W_1 - (\lambda_1(t_1) - (t_1 - t_0)\Delta) + (t_1 - t_0)\gamma_1\Delta \\ &= W_1 - \lambda_1(t_1) + (t_1 - t_0)(1 + \gamma_1)\Delta\end{aligned} \tag{B.4}$$

Since t_0 is the time when the aggregate sending rates is equal to the bottleneck capacity, it is reasonable to say that the bottleneck queue size is zero at time t_0. Since at t_1, the queue is full, the queue is filled by sending rate increments during $[t_0, t_1]$. Recall that the total sending rate grows by an aveage rate of $\Delta + \Delta_1^o = (1 + \gamma_1)\Delta$ per unit time. Even though it is possible that Δ_1^o can be negative at times, expected growth rate of the flows will be positive paticularly before time t_1 due to available buffer space which prevents loss. So, we assume that the aggregate flow growth rate $(1+\gamma_1)\Delta$ is positive during $[t_0, t_1]$. Hence, the following equality can be written as the relationship between the buffer size and the flows growth rate:

$$\frac{1}{2}(t_1 - t_0)^2(1 + \gamma_1)\Delta = Q_1 \Rightarrow t_1 - t_0 = \sqrt{\frac{2Q_1}{(1 + \gamma_1)\Delta}}$$

Together with (B.4), we can write:

$$\lambda_1^o(t_1) = W_1 - \lambda_1(t_1) + \sqrt{2\Delta Q_1(1 + \gamma_1)} \tag{B.5}$$

Assuming all flows going through the bottleneck have the same priority and the delay from the bottleneck to the CR is ignorable, since at time t_1 the bottleneck is working at its full load, we know that the following equality holds:

$$\begin{aligned}\omega_1(t_1) &= \frac{\lambda_1(t_1)}{\lambda_1(t_1) + \lambda_1^o(t_1)} W_1 \\ &\overset{(B.5)}{=} \frac{\lambda_1(t_1)}{1 + \frac{1}{W_1}\sqrt{2\Delta Q_1(1 + \gamma_1)}}\end{aligned} \tag{B.6}$$

On any other path j ($j = 2 \ldots N$), since the ERMCC source ignores the congestion indications on this path (Figure B.2), the sending rate of the ERMCC

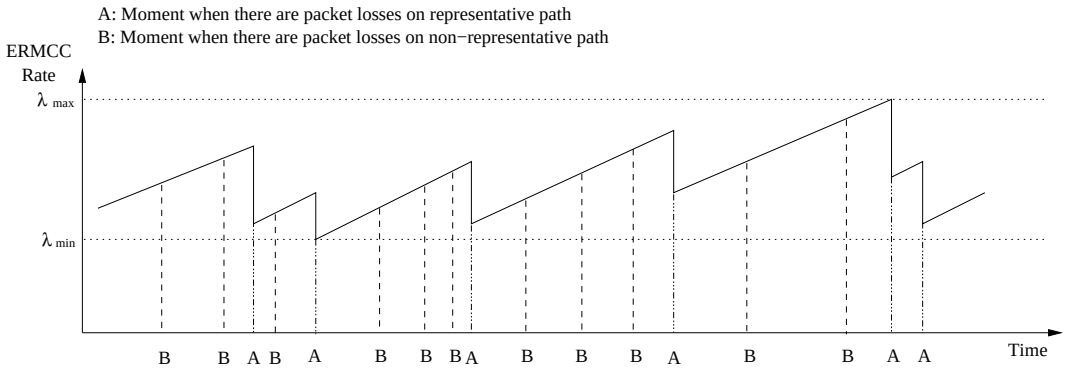


Fig. B.2. The ERMCC source only considers the congestions on the representative path for rate adaptation.

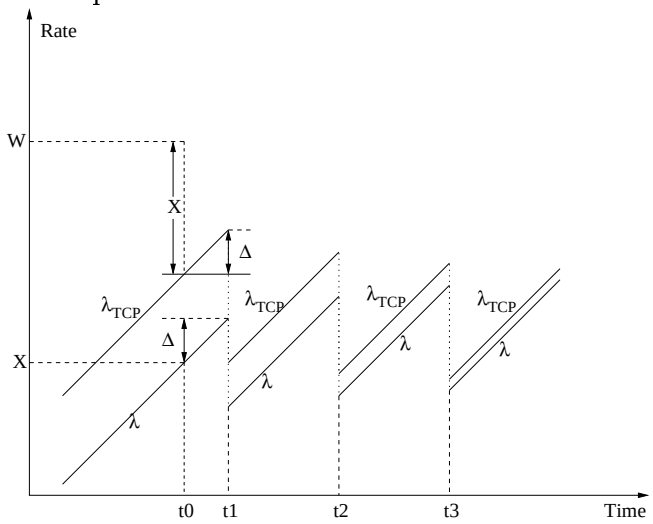


Fig. B.3. Evolution of the sending rates for TCP and ERMCC flows.

flow still grows by Δ per unit time. With t_1 of the same meaning as before, according to a derivation similar to that above, we have:

$$\omega_j(t_1) = \frac{\lambda_j(t_1)}{1 + \frac{1}{W_j}\sqrt{2\Delta Q_j(1+\gamma_j)}} \tag{B.7}$$

Consider $\lambda_i(t_1)$ $(i = 1 \ldots N)$. Assume that the sending rate of the ERMCC flow varies between λ_{min} and λ_{max} (Figure B.2), then $\lambda_i(t_1)$ is a sample value of a random variable Λ_i with sample space as $[\lambda_{min}, \lambda_{max}]$. Since the source's sending rate λ is the same for all paths, the average value of λ_i the observed sending rate at the bottleneck on path i will be the same for all paths $i = 1..N$, i.e. $E[\Lambda_i] = E[\Lambda_j],\ i \neq j$.

Again assuming that the delay between the bottleneck and the receiver on paths is ignorable, we can write $\Omega_i(t_1) = E[\omega_i(t_1)]$, which means the TRAC measured at receiver i can be written in terms of the output rate obtained in (B.7). Assuming that W_i, Q_i, s are constant, and that γ_i and $\widehat{RTT}$ in steady state and have small deviations and thus can be treated as constant, we have

$$\omega_i = \frac{\Lambda_i}{1 + \frac{1}{W_i}\sqrt{2\Delta Q_i(1+\gamma_i)}} \tag{B.8}$$

$$\Omega_i = \frac{E[\Lambda_i]}{1 + \frac{1}{W_i}\sqrt{2\Delta Q_i(1+\gamma_i)}} \tag{B.9}$$

B.2 Capability of Tracking The Slowest Receiver

In the previous subsection, we derived a steady-state formulation for the TRAC. Like we did in the previous subsection, let path 1 be the Representative Path which has the CR as its receiver. As designed in ERMCC, for $j = 2 \ldots N$, only upon detection of $\mu_j < \mu_1$ will receiver j send a congestion indication in terms of feedback packets back to the source, which will then update the CR to receiver j. We can write an expression for the average TRAC based on the expression of Ω_i in (B.9) as follows:

$$\begin{aligned}\mu_i &= \Phi(\mu_i, \Omega_i, \alpha) \\ &= \frac{\Phi(E[\Lambda_j], \alpha)}{1 + \frac{1}{W_i}\sqrt{2\Delta Q_i(1+\gamma_i)}}\end{aligned} \tag{B.10}$$

This means that the average TRAC will be dependent on moving average of the source's sending rate. The condition of CR change is:

$$\begin{aligned}&\mu_j < \mu_1 \\ \Leftrightarrow\ &\frac{\Phi(E[\Lambda_j], \alpha)}{1 + \frac{1}{W_j}\sqrt{2\Delta Q_j(1+\gamma_j)}} < \frac{\Phi(E[\Lambda_1], \alpha)}{1 + \frac{1}{W_1}\sqrt{2\Delta Q_1(1+\gamma_1)}} \\ \Leftrightarrow\ &\frac{W_j}{\sqrt{Q_j(1+\gamma_j)}} < \frac{W_1}{\sqrt{Q_1(1+\gamma_1)}} \\ &\text{since } E[\Lambda_j] = E[\Lambda_1]\end{aligned} \tag{B.11}$$

We can see that $W_i/\sqrt{Q_i(1+\gamma_i)}$ $(i = 1 \ldots N)$ indicates the degree of congestion on the bottleneck of path i. In fact, if the bottleneck has less bandwidth (i.e. W_i is smaller) or larger buffer (i.e. Q_i is larger), then $W_i/\sqrt{Q_i(1+\gamma_i)}$ has a lower value. Also, if more flows are sharing a bottleneck, the sum of their

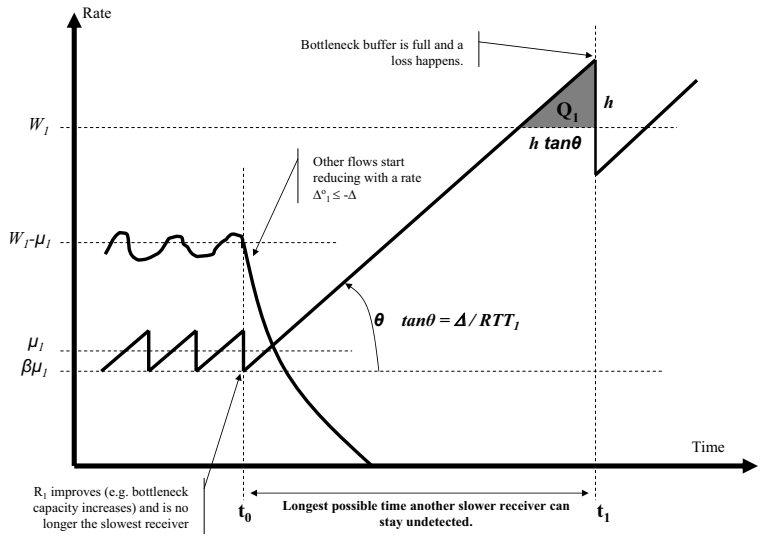


Fig. B.4. Longest possible time the ERMCC flow cannot detect a slower receiver.

per-unit-time rate increments Δ_i is higher, $\gamma_i = \Delta_i/\Delta$ is then larger, which in turn also makes $W_i/\sqrt{Q_i(1+\gamma_i)}$ lower. Therefore, (B.11) actually shows that as long as a non-representative path (path j) experiences a more serious congestion than the representative path (path 1) does, the receiver behind path j will see lower average TRAC μ_j, and will send congestion indications back to the source, making the source change CR. This steady-state analysis shows that *an ERMCC flow tracks the slowest receiver.*

However, one issue is to analyze the transient behavior, i.e. an analysis of how long it takes the ERMCC flow to detect and update the CR to a receiver which has just become the slowest. We analyze the transient behavior by condisering the two possible cases as we identfied earlier in Section 3.2.1:

- **Case I:** R_1 *improves.* The current CR R_1 might become faster just so that another flow R_j, $j = 2..N$ is slower than R_1. This means R_j is supposed to be the new CR.
- **Case II:** R_j *worsens.* The curent CR stays the same, but another flow R_j, $j = 2..N$ becomes slower than R_1. This means R_j is supposed to be the new CR.

Lemma 1 *For Case I, ERMCC's time to detect the new slowest receiver is bounded by* $\sqrt{2Q_1/s} + W_1/\Delta$ *round trip times.*

PROOF. Figure B.4 depicts the scenario for the longest possible time a slower flow can stay undetected by ERMCC when the current CR R_1 becomes faster than it was before. The figure plots the rate of R_1 in comparison to the aggregate rate of the other flows on the bottleneck on path 1. Assume that at time t_0, the current CR R_1 improves because of a local reason such as

an increase in the available bottleneck capacity or more availability of computation power at R_1. Let there be another flow R_j which is slower than what the current R_1 has become at time t_0. This means, starting at time t_0, R_j is the new slowest receiver.

The longest possible time period of R_j staying undetected happens when two things happen at the same instant t_0: (i) R_1 has just generated a congestion indication at time t_0, and (ii) all other flows using the same bottleneck start reducing their sending rate by $\Delta_1^o \leq -\Delta$.

After the instant t_0, R_1 will reach the total bottleneck capacity W_1 and fill up the available buffer Q_1. Then, a loss will occur and the undetected slowest receiver R_j will be uncovered because of the communication between the source and the receivers.

In Figure B.4, let the height of the shaded triangle be h. Since that triangle has an area equal to the buffer size Q_1, the following holds:

$$h = \sqrt{\frac{2\Delta Q_1}{\widehat{RTT}}}$$

Similarly, from the large triangle, we can write the following:

$$\begin{aligned} t_1 - t_0 &= \frac{\widehat{RTT}}{\Delta}\left[\sqrt{\frac{2\Delta Q_1}{\widehat{RTT}}} + W_1 - \beta\mu_1\right] \\ &= \widehat{RTT}\left[\sqrt{\frac{2Q_1}{s}} + \frac{W_1 - \beta\mu_1}{\Delta}\right] \end{aligned} \tag{B.12}$$

By eliminating $\beta\mu_1$ from (B.12),we can conclude that

$$t_1 - t_0 < \widehat{RTT}[\sqrt{2Q_1/s} + W_1/\Delta]$$

□

Lemma 2 *For Case II, ERMCC's time to detect the new slowest receiver is bounded by* $2\max(RTT_i)$, $i = 1..N$.

PROOF. When another receiver R_j becomes slower than the current CR R_1, it will send a congestion indication to the source, which will take one round trip time, i.e. RTT_j. Then, the source will inform all the receivers about the new slowest receiver. In the worst case, R_j can be the receiver with the longest RTT, and it may take another RTT due to packetization effects. So, the longest time to detect the new slowest receiver will be $2\max(RTT_i)$. □

Theorem 1 *ERMCC can detect any slowest receiver which stays to be the slowest longer than* $\max(2, \sqrt{2Q_1/s} + W_1/\Delta)RTT_{max}$.

PROOF. Proof follows from Lemma 1 and Lemma 2. □

B.3 TCP-Friendliness on Representative Path

Assuming that their RTT estimations and packet sizes are the same, we now show that an ERMCC flow is friendly to a TCP flow on the representative path, i.e. they get approximately equal share of the bottleneck bandwidth. More specifically, we want to show that, with proper choice of rate reduction factor β for ERMCC, $\lambda(t)/\lambda^{TCP}(t)$ oscillates around 1, where $\lambda(t)$ and $\lambda_{TCP}(t)$ denote the sending rates of the ERMCC flow and the TCP flow at time t respectively. Those two flows are assumed to be the only flows on the representative path. A sample of the rate evolution is given in Figure B.3.

Like other TCP throughput analysis papers [10] [11] have done, our analysis focuses only on TCP's congestion avoidance behavior. During congestion avoidance period, when without packet losses, a TCP source increases its congestion window by $1/cwnd$ packet upon the receipt of per ACK, where *cwnd* is the current congestion window size. A TCP source transmits all the packets in its congestion window in one RTT, therefore, the window grows by 1 packet per RTT, [12] which corresponds to the fact that its sending rate is increased by s/RTT per RTT, where s is the packet size. An ERMCC source increases its sending rate at the same pace, as covered in scheme description. At packet loss, a TCP source will reduce its congestion window by half, which is equivalent to cutting its sending rate by half.

Assume that congestion is the only reason for packet losses. It is obvious that packet losses can occur only if $\lambda_{TCP}(t) + \lambda(t) \geq W$. Suppose some packets are lost and both flows reduce their transmission rates at t_1 (Figure B.3). Before the losses, since both $\lambda_{TCP}(t)$ and $\lambda(t)$ keep increasing, there must be a moment t_0 when $\lambda_{TCP}(t_0) + \lambda(t_0) = W$. In short, let $\lambda(t_0) = X$, then $\lambda_{TCP}(t_0) = W - X$. For the first step of analysis, we will show that with appropriate β,

$$\begin{cases} X < W - X \Rightarrow X/(W-X) < \lambda(t_1)/\lambda_{TCP}(t_1) \\ \\ X > W - X \Rightarrow X/(W-X) > \lambda(t_1)/\lambda_{TCP}(t_1) \end{cases} \tag{B.13}$$

[12] We assume that a TCP receiver sends an ACK for every received packet.

Let the moment just before the rate reduction at t_1 be t_1'. Since both of the flows share the same path, we can assume that they detect packet losses and reduce transmission rates approximately at the same time. For the TCP flow, suppose that at t_1', its transmission rate has been increased by d since t_0, i.e.

$$\lambda_{TCP}(t_1') = W - X + d$$

After a reduction by half,

$$\lambda_{TCP}(t_1) = \frac{\lambda_{TCP}(t_1')}{2} = \frac{W - X + d}{2}$$

Since the ERMCC flow increases its rate at the same pace, we have,

$$\lambda(t_1') = X + d$$

Assume that both flows have the same priority, i.e. their packets are forwarded by the bottleneck with the same probability. In consequence, at t_1', the ERMCC CR sees an approximate receiving rate of

$$\frac{\lambda(t_1')}{\lambda(t_1') + \lambda_{TCP}(t_1')} W = \frac{X + d}{W + 2d} W$$

According to the rate adaptation policy of ERMCC,

$$\lambda(t_1) = \beta \frac{X + d}{W + 2d} W$$

Therefore,

$$\frac{\lambda(t_1)}{\lambda_{TCP}(t_1)} = \beta \frac{X + d}{W + 2d} W \Big/ \frac{W - X + d}{2}$$

Now let's compare $X/(W - X)$ and $\lambda(t_1)/\lambda_{TCP}(t_1)$.

$$\begin{aligned} & \frac{X}{W - X} - \frac{\lambda(t_1)}{\lambda_{TCP}(t_1)} \\ = & \frac{2}{W - X + d} \\ & \cdot \left[\left(\frac{1}{2} - \frac{\beta W}{W + 2d} \right) X + \left(\frac{X}{2(W - X)} - \frac{\beta W}{W + 2d} \right) d \right] \end{aligned} \tag{B.14}$$

Since $W > X$ and $d \geq 0$, $2/(W - X + d) > 0$, and the positivity of (B.14) is decided by its second factor between the square brackets. If we choose a value for β carefully so that,

$$\beta = \frac{1}{2} \frac{W + 2d}{W}$$

then the second factor of (B.14) becomes:

$$0 \cdot X + \frac{d}{2}\left(\frac{X}{W-X} - 1\right) = \frac{d}{2}\left(\frac{X}{W-X} - 1\right) \tag{B.15}$$

It is easily seen that, if $X > W - X$, (B.15) > 0 so that (B.14) > 0; while if $X < W - X$, (B.15) < 0 so that (B.14) < 0. That is exactly what we want for (B.13) to hold.

With (B.13) established, we can extend (B.13) to all time instants t_i, $i = 1 \ldots \infty$ when both the TCP flow and the ERMCC flow reduce their rates (Figure B.3). Let sending rates after reduction be $\lambda_{TCP}(t_i)$ and $\lambda(t_i)$ respectively for the TCP and ERMCC flows. Also assume that t_{i-1} is the last moment before t_i so that $\lambda_{TCP}(t_{i-1}) + \lambda(t_{i-1}) = W$. We can write the following relationship:

$$\begin{cases} \lambda(t_{i-1}) < \lambda_{TCP}(t_{i-1}) \Rightarrow \frac{\lambda(t_{i-1})}{\lambda_{TCP}(t_{i-1})} < \frac{\lambda(t_i)}{\lambda_{TCP}(t_i)} \\ \\ \lambda(t_{i-1}) > \lambda_{TCP}(t_{i-1}) \Rightarrow \frac{\lambda(t_{i-1})}{\lambda_{TCP}(t_{i-1})} > \frac{\lambda(t_i)}{\lambda_{TCP}(t_i)} \end{cases} \tag{B.16}$$

As the result, if the ERMCC flow rate is less than that of the TCP flow, it will grow until it exceeds the latter; likewise, if the ERMCC flow rate is more, it will get less and less until it is below the TCP flow rate. Hence, $\lambda(t)/\lambda_{TCP}(t)$ oscillates around 1, which means ERMCC is TCP-friendly as long as the rate reduction factor β is properly chosen as:

$$\beta = \frac{1}{2}\frac{W + 2d}{W}$$

Since $d \geq 0$, β needs to have a value greater than 0.5. Considering the fact that TCP uses ACKs to measure RTTs, it can have lower RTT estimation than that of ERMCC which uses NAKs for this purpose, as we discussed in Section 3.2.4. Thus, TCP can increase sending rate faster than ERMCC. To compensate this, ERMCC at packet losses can reduce its transmission rate by less using larger value of β. In our implementation, we use a value of 0.75 and it works fine in simulations.

B.4 Immunity to Drop-to-zero Problem

The cause of drop-to-zero problem is the asynchronous packet losses on multiple paths. If a multicast source reduces the transmission rate too much on the losses, the rate will stay very low or even converge to zero. Since the source in ERMCC adapts the transmission rate according to the congestion on one

single path while ignoring that on all others, there will be no drop-to-zero problem for ERMCC.

More specifically, if a receiver other than the current CR sees a packet loss rate lower or equal to that by CR, it will not send feedbacks to the source. The source will not see any feedbacks from it, and thus will not reduce the transmission rate. Even if the source gets feedbacks from different receivers because there is a change of the most congested bottleneck, once it chooses a receiver as the new CR after a very short period of time (several RTTs), it will ignore feedbacks from all other receivers. Consequently, *due to its usage of single receiver as the CR, ERMCC is immune to drop-to-zero.*

B.5 Effectiveness of Feedback Suppression

Without support from internal nodes, which is the situation that we assume for practical purposes, most multicast feedback suppression schemes (e.g. [18], [19], [3], [12], [20]) use random timers for delaying receivers' feedback before sending them. This suppresses some of the feedback, however, it also causes some latency – even though small – to the needed feedback which may bring performance penalty. Since our feedback suppression is not based on timers, it does not suffer from this problem. Also, there is no need to know or estimate the total number of receivers like in [12].

Besides the crucial benefit above, we are also going to show below that, in ERMCC, the total number of feedbacks (i.e. congestion indications) sent to the source by all receivers is independent of the total number of receivers. Instead, it depends on (i) the switching frequency of the most congested bottleneck, (ii) the number of receivers behind the new most congested bottleneck, and (iii) the minimum RTT between the receivers behind the new bottleneck and the source. For convenience, we use the acronym *MCB* for *most congested bottleneck* in the following discussion.

We assume that there is only one MCB at any moment [13] . Condider the case when a new MCB is realized. Let M be the total number of receivers behind the new MCB, and R_i be the receiver behind the new MCB, where $i = 1 \ldots M$. Also, let RTT_i^f be the forward (downstream) latency (part of RTT_i) towards the receiver R_i, and RTT_{min} be the minimum RTT among all receivers behind the new MCB, i.e. $RTT_{min} = \min_{i=1..M} RTT_i$. Finally, let p be the packet loss rate experienced by the receivers behind the new MCB [14] .

[13] There can certainly be multiple bottlenecks which have similar degree of congestion and are all most congested. However, the discussion still holds.

[14] Since the receivers involved here are all behind MCB, we can assume that they see the same degree of congestion and thus the same packet loss rates.

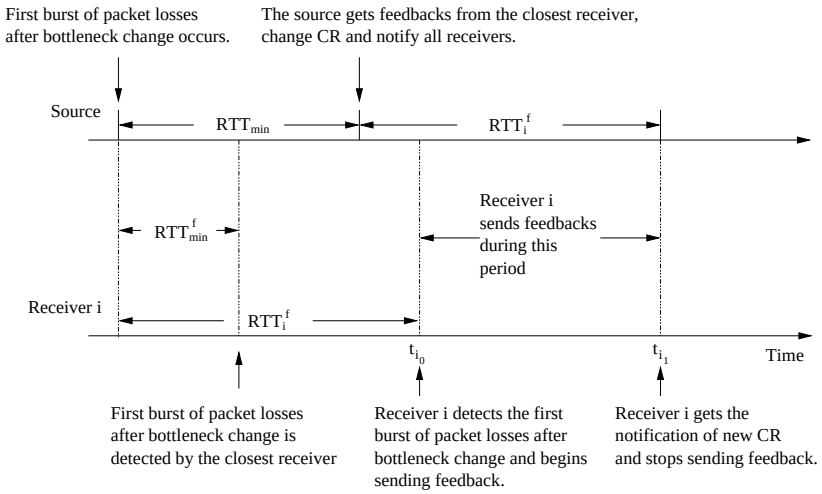


Fig. B.5. Feedback suppression.

Whenever there is a new MCB, only those receivers behind the new MCB will send congestion indications back to the source. After one of them is chosen as the new CR, all of them except one will stop sending the feedbacks. More specifically, the source will first see the feedbacks from the receiver with RTT_{min}, then change CR and tell all receivers about the change. For any R_i except the new CR, the duration of sending feedbacks is between the moment t_{i_0} when they first detect packet loss after bottleneck change and the moment t_{i_1} when they know about the new CR. According to Figure B.5, $t_{i_1} - t_{i_0} = RTT_i^f + RTT_{min} - RTT_i^f = RTT_{min}$. Therefore, before a new CR is decided, (i) the number of feedbacks sent from this receiver R_i is $p\bar{\lambda}RTT_{min}$, and (ii) the total number of feedbacks sent by all receivers behind the new MCB is $\bar{\lambda}pM \cdot RTT_{min}$.

Once a new CR is decided, only one the new CR, will send feedbacks. Let's call the period between two successive MCB switchings as *MCBSP (MCB Switching Period)*. During a MCBSP of length $t \geq RTT_{min}$, the total number of feedbacks sent to the source is:

$$\bar{\lambda}pM \cdot RTT_{min} + \bar{\lambda}p(t - RTT_{min}) = \bar{\lambda}p(t + (M-1)RTT_{min})$$

Assume that MCB switching times follows a Poisson distribution with an average inter-arrival time of $1/\delta$. In practice, MCB switching will not occur too frequently and it is reasonable to assume that MCBSPs will be larger than RTT time-scale, i.e. $1/\delta \geq RTT_{min}$. For a multicast session with average duration of T, the total number of feedbacks transmitted will approximately be [15]:

$$\delta T \cdot \bar{\lambda}p\left(\frac{1}{\delta} + (M-1)RTT_{min}\right) \tag{B.17}$$

[15] Finally, we would like to note that due to measurement errors in practice, the total number of feedbacks sent can be a little higher/less than what we have derived here. However, the difference will not be significant.

We can see that, for a certain T,

$$\begin{aligned} &\lim_{\delta \to 0} \delta T \cdot \bar{\lambda} p \left(\frac{1}{\delta} + (M-1) RTT_{min} \right) \\ &= \bar{\lambda} pT + \lim_{\delta \to 0} \delta \bar{\lambda} pT (M-1) RTT_{min} = \bar{\lambda} pT \end{aligned} \tag{B.18}$$

That means, if there is no MCB switching, the total number of feedbacks sent is approximately equal to the number of feedbacks sent from a single receiver behind the MCB. In other words, *if the MCB does not change during a multicast session, the volume of feedback messages is on the same level of a unicast session.*

Also, from (B.17), we find that the total number of transmitted feedbacks is independent of the total number of receivers in a multicast session [16]. It depends on how fast MCB switches (i.e. $1/\delta$), the amount of receivers behind the new MCB (i.e. M), and the smallest RTT between those receivers and the source (i.e. RTT_{min}). Usually, MCB switches only once in many RTT_{min}s, and the amount of receivers behind the new MCB is much less than the overall number N. Moreover, RTT_{min} is almost a negligible duration. Consequently, our feedback suppression mechanism is effective.

[16] Note that M is (B.17) is not the total number of receivers but the number of receivers behind the MCB