Available online at www.sciencedirect.com
SCIENCE @ DIRECT®
Computer Networks 50 (2006) 2614–2640
Computer Networks
www.elsevier.com/locate/comnet

Explicit rate multicast congestion control

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

[a] Rensselaer Polytechnic Institute, Electrical Computer and Systems Engineering Department, 110 8th Street, Troy, NY 12180, United States
[b] Howard University, Department of Systems and Computer Science, 2300 6th Street NW, Washington, DC 20059, United States

Received 8 April 2005; received in revised form 25 September 2005; accepted 26 September 2005
Available online 2 November 2005

Responsible Editor: E. Chong

Abstract

In this article, we propose a new single-rate end-to-end multicast congestion control scheme called *Explicit Rate Multicast Congestion Control* (*ERMCC*) based on a new metric, *TRAC* (*Throughput Rate At Congestion*). ERMCC can be implemented only at the source and the receivers of the multicast tree. 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. ERMCC does not suffer from the *drop-to-zero* problem and is very effective with feedback suppression (achieves 95% suppression). Furthermore, with proper adjustment of its rate adaptation parameters, ERMCC achieves TCP friendliness only on the path to the slowest receiver. 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.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Multicast; Congestion control; Single-rate; Drop-to-zero; TCP friendliness; Feedback suppression; Throughput rate at congestion (TRAC)

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

* Corresponding author. Tel.: +1 518 276 6823; fax: +1 925 888 2167.
E-mail addresses: lij@scs.howard.edu (J. Li), yuksem@ecse.rpi.edu (M. Yuksel), shivkuma@ecse.rpi.edu (S. Kalyanaraman).

1389-1286/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2005.09.026

capabilities, also does not introduce high processing load to them. However, single-rate mechanisms underutilize the network and slow receivers can significantly reduce the quality perceived by faster receivers in the multicast group. Multi-rate approaches protect fast receivers from extremely slow receivers, by excluding them from the group, especially if the application has minimum bandwidth or maximum delay requirements. Although single-rate 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. Recently, it has been shown that (e.g., GMCC [9] and SMCC [10]) that one can elegantly compose multi-rate schemes from single-rate schemes. Thus, single-rate schemes can be used as building blocks for developing multi-rate schemes, which shows the need to study single-rate schemes.

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 provide a baseline ERMCC scheme which can be used for both reliable and unreliable data applications. The baseline ERMCC scheme we present in this paper can be used for real-time loss-tolerant applications such as multimedia streaming. Also, this baseline scheme can be extended for reliable data applications by including a combination of reliability mechanisms [11] (e.g., proactive/reactive forward error correction (FEC), digital-fountain style FEC [12], ARQ, local recovery, PGM-like NAK aggregation [13,2], and application layer reliability such as Application-Layer Framing [14]).

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 significantly 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. We enlisted symbolic representations in Table 1 to ease the understanding

[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.

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

of the scheme. In Fig. 1(a), let us 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 indications* (CIs), because they are sent only when congestion is detected due to packet loss. As shown in Fig. 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 $1 \ldots k$ behind that path will see average TRACs lower than that of the current CR (i.e., $\forall i = 1 \ldots k, \mu_i(t_1) < \hat{\mu}(t_0) - \hat{\sigma}(t_0)$), and will send feedbacks as shown in Fig. 1(b). As the result, one of them will be chosen as the new CR.

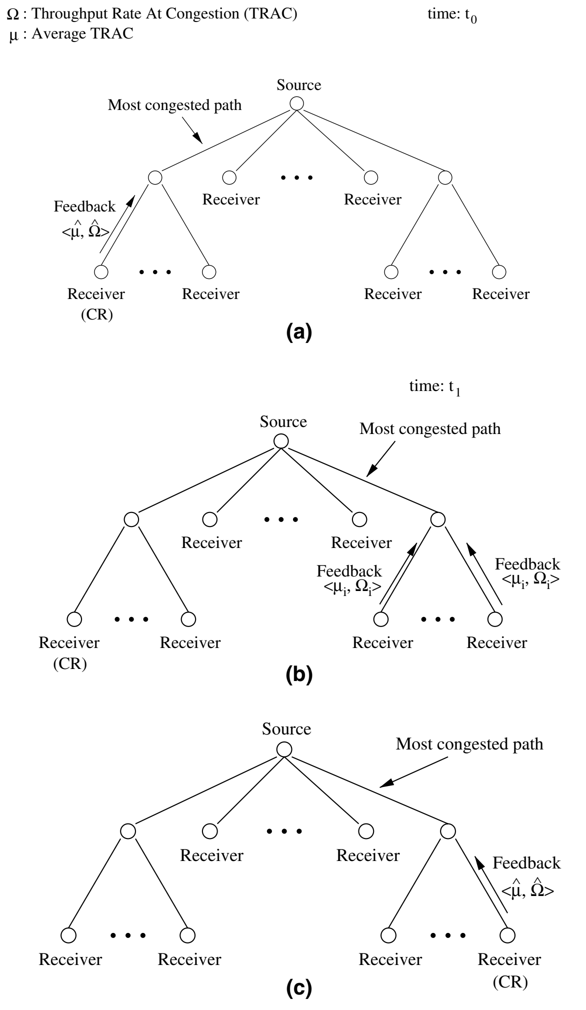

Fig. 1. Example operation of ERMCC.

After that, again, other receivers will suppress their feedback as shown in Fig. 1(c).

1.2. Key contributions

ERMCC introduces a novel method of using *explicit rate* feedback [15] 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. ERMCC also is immune to *the drop-to-zero problem*. With proper adjustment of parameters for an

[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.

AIMD-like rate adaptation technique, ERMCC achieves approximate *TCP-friendliness* on the representative path (i.e., the path between the source and the slowest receiver). In addition, only the 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:

- We propose 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.
- We analyze the scheme performance theoretically.
- By comparison in simulation with PGMCC [2] and TFMCC [3], we show that ERMCC achieves better performance under most situations.
- We present the test results of ERMCC implementation on a real testbed system in Emulab [16].
- We also present large-scale simulation test results of ERMCC.

2. Related work

Multicast research work can be classified into two main groups as: (i) the ones with end-to-end operation without requiring any network assistance (i.e., help of intermediate network routers'), and (ii) the ones that require network assistance. Another type of classification can be made in terms of the variability of source's sending rate in the multicast session: (i) single-rate schemes, and (ii) multi-rate schemes. Our work belongs to the categories of end-to-end and single-rate multicast schemes. Based on the latter categorization of multicast proposals, we survey the related work in the subsections below.

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 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 [17,18] 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-side RTT measurement. TFMCC comes with feedback suppression which is an enhanced version of [19] 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

In multi-rate multicast congestion control, receivers may obtain different throughput rates. Ideally, data can reach each receiver at the rate that matches the condition of the path between the sender and the particular receiver.

To realize such effects, it is a commonly accepted approach to use multiple simultaneous multicast groups (known as "layers") for transmission. Based on a set of metric, each receiver independently and dynamically joins and leaves these layers during the course of a session. As the result, the total throughput of joined layers as the session throughput varies from receiver to receiver.

In early multi-rate schemes, such as RLM [20], RLC [21], PLM [22], RLS [23] and MSC [24], the transmission rates of layers are fixed. Receivers join layers accumulatively, and leave in the reverse order. The throughput adaptation therefore totally depends on receivers' join and leave actions. Some following multi-rate schemes, such as FLID-DL [25], Fine-Grained Layered Multicast [6] and STAIR [7], still use fixed layer sending rates, but are more careful about the join and leave operations by receivers. By carefully designating the sending rates of layers and the order of join and leave, receivers emulate the increase and decrease of throughput more smoothly and thus achieve better performance.

Other more recent multi-rate schemes, such as MLDA [26], HALM [27], SMCC [10] and our work GMCC [9], allow the source to adjust the layer sending rates. SMCC and GMCC in particular provide a way of composing multi-rate multicasting paradigm at by using single-rate schemes as building blocks. The capability of adjustment at both sides (sender and receiver) allows for more adaptability to the network condition and yields better results.

Obviously, multi-rate schemes are more suitable for heterogeneous environments where they can utilize bandwidth more efficiently. Nonetheless, such schemes are complicated in their design. They are *closely coupled* with IGMP due to the needs of join and leave. Since the IGMP operations have routers involved, the time required to accomplish adaptations is relatively long, which leads to the slackness of response to congestion. To make things worse, if aggregated multicast trees [28] are used, trees may not be pruned accordingly when leave operations are requested, resulting in irresponsiveness to congestion.

Additionally, most of the existing multi-rate schemes are bound to routing as well, assuming that all layers in a multicast session use the same multicast tree (so that the join and leave operations by a receiver can lead to the increase and decrease of traffic volume on the path between the sender and the receiver). Unfortunately, in some cases, it is possible for different layers to use different routes [29].

In comparison, single-rate schemes are relatively simple and better understood. They fit better for relatively homogeneous environments and other situations where efficient bandwidth utilization is not a big concern. Moreover, single-rate schemes can be used as building blocks in multi-rate schemes such as aforementioned SMCC [10] and GMCC [9]. Therefore, it is still valuable to study single-rate schemes.

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): $\mu(t)$, $\Omega(t)$, $\omega(t)$

ERMCC receivers measure and maintain TRAC, which later gets fed back to the source for rate adaptation and CR selection. Receivers measure the TRAC at multiple time-scales and employ statistical measures like standard deviation to smooth the TRAC measurements.

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 s for all cases in this paper. It is important to note here that optimal value of Δt depends on (i) burstiness of the cross traffic for the multicast tree and (ii) sensitivity of the multicast application to loss. Large Δt may cause losses in case of dramatic changes in the background traffic over the representative path to the CR. When the background traffic suddenly goes away on the representative path, the CR might then send several feedbacks to the source. This may cause the source to maintain or increase its sending rate until the end of TRAC measurement period, Δt. Such a behavior may cause losses at the newer slowest receiver if there happens to be a newer slowest receiver than the current CR. To avoid such possibilities, it is safer to tune Δt to not more than a few RTTs. In our experiments for evaluating ERMCC in this paper, we set Δt to a relatively large value of 1 s due to two reasons: (i) to make sure that Δt is larger than the maximum RTT in the topology, and (ii) to make a conservative evaluation of the scheme. Therefore, setting Δt to smaller values (e.g., maximum RTT of the topology) will improve the performance of ERMCC.

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

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

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

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

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 Fig. 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

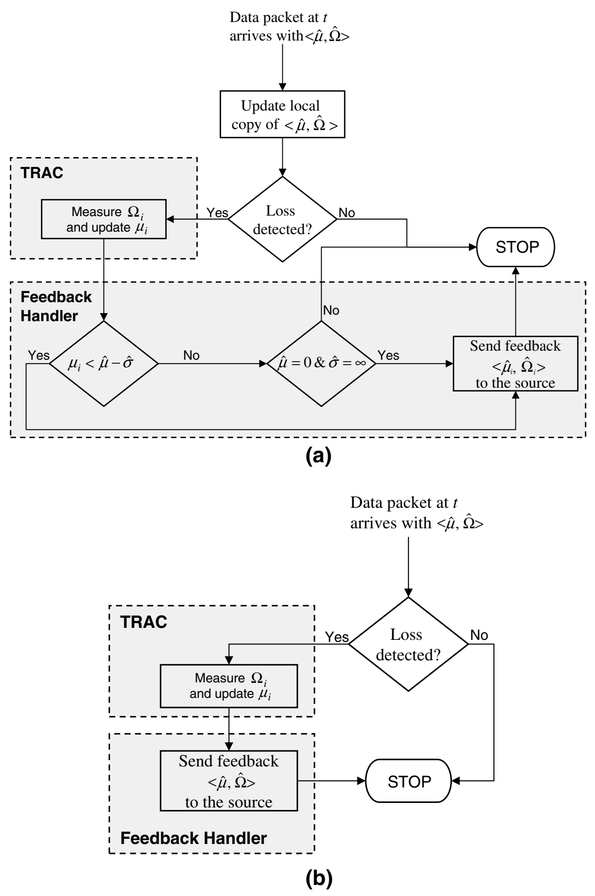
Data packet at t arrives with $<\hat{\mu},\hat{\Omega}>$
Update local copy of $<\hat{\mu},\hat{\Omega}>$
TRAC
Measure Ω_i and update μ_i
Loss detected?
Yes
No
STOP
Feedback Handler
$\mu_i < \hat{\mu} - \hat{\sigma}$
$\hat{\mu} = 0 \& \hat{\sigma} = \infty$
Send feedback $<\hat{\mu}, \hat{\Omega}_i>$ to the source
(a)
Data packet at t arrives with $<\hat{\mu}, \hat{\Omega}>$
TRAC
Loss detected?
Measure Ω_i and update μ_i
Send feedback $<\hat{\mu}, \hat{\Omega}>$ to the source
STOP
Feedback Handler
(b)

Fig. 2. Operations at ERMCC receiver: (a) non-CR receiver i and (b) CR receiver.

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 Fig. 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 Fig. 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. ERMCC receivers achieve very effective suppression of feedbacks by applying smoothing techniques like EWMA and statistical measures like the standard deviation.

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 non-CR 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 $\mu_i(t) \leqslant \hat{\mu}(t)$ to be conservative and keep the choice of 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 (Fig. 6). To remedy the possibility 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 Fig. 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 [18,17] which requires per receiver RTT and packet loss rate.

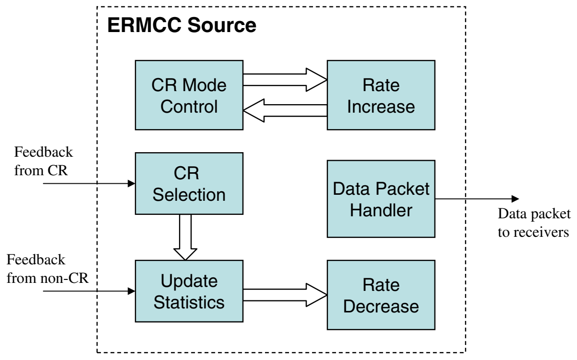

Fig. 3. Source operations as a block diagram.

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 Fig. 2(a) and CR Selection part of Fig. 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. 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 Fig. 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 Fig. 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 Fig. 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 loaded, 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 Fig. 4.

To see this mode control process on a timeline, let us look at Fig. 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 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., $T_{\max} = E[T] + 8T_{\sigma}$.

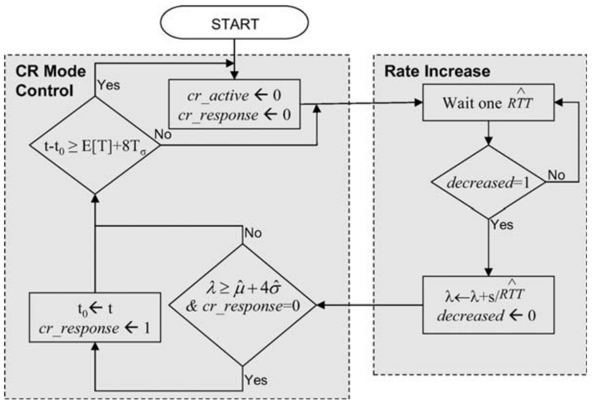
START
CR Mode Control
Yes
$cr\_active \leftarrow 0$
$cr\_response \leftarrow 0$
No
$t\text{-}t_0 \geq E[T]+8T_\sigma$
No
$\lambda \geq \hat{\mu}+4\hat{\sigma}$ & $cr\_response=0$
$t_0 \leftarrow t$
$cr\_response \leftarrow 1$
Yes
Rate Increase
Wait one $\widehat{RTT}$
$decreased=1$
No
Yes
$\lambda \leftarrow \lambda + s/\widehat{RTT}$
$decreased \leftarrow 0$

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*.

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.

ERMCC is a *rate-based* scheme, using the policy of additive increase and multiplicative decrease (AIMD). As shown in Rate Increase part of Fig. 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 min-

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

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

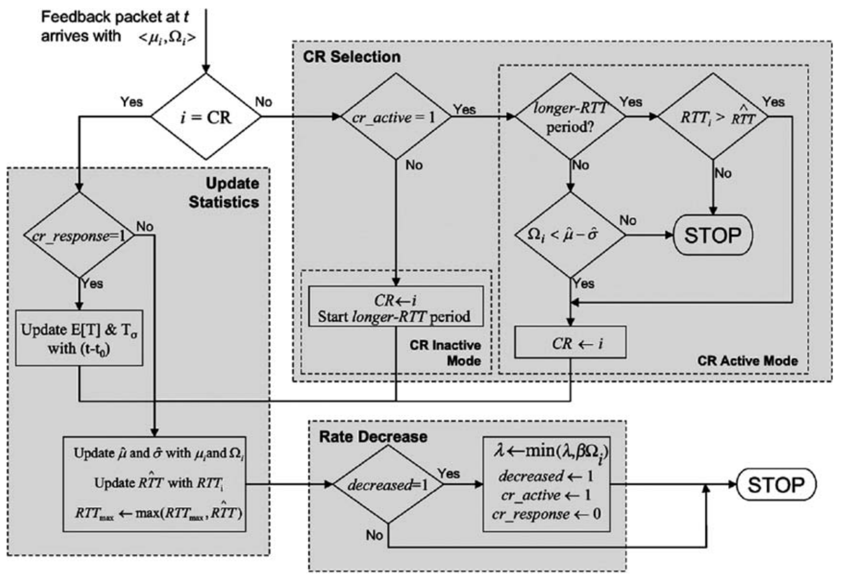

Fig. 5. Operations that take place when a feedback packet from receiver i arrives at the source.

imum 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 Fig. 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., $t_0 < t \leqslant t_1$.

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 Fig. 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 queuing delay and thus is still the upper bound. On the other hand, ACKs as those in TCP may or may not include bottleneck queuing 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 Fig. 5, *upon receipt of a feedback* from receiver i, the source updates $\widehat{RTT}$

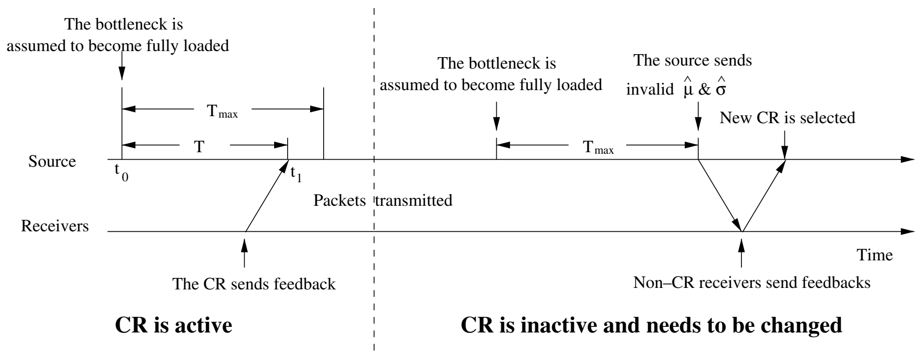
The bottleneck is assumed to become fully loaded
T max
T
Source
t 0
t 1
Packets transmitted
Receivers
The CR sends feedback
CR is active
The bottleneck is assumed to become fully loaded
The source sends invalid μ̂ & σ̂
New CR is selected
T max
Time
Non–CR receivers send feedbacks
CR is inactive and needs to be changed

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

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 Fig. 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 Fig. 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,

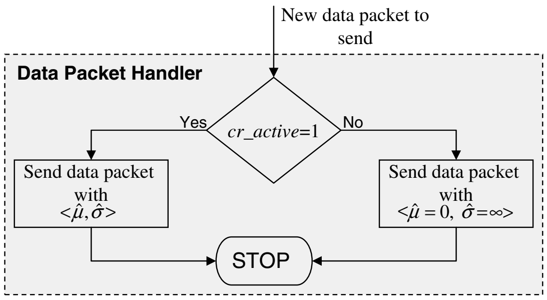
New data packet to send
Data Packet Handler
cr_active=1
Yes
No
Send data packet with <μ̂,σ̂>
Send data packet with <μ̂ = 0, σ̂=∞>
STOP

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

ERMCC source attaches them to the data packets. As shown in Fig. 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 pseudocode of ERMCC's algorithm is provided in Appendix A for reference. The code for *ns*-2 and Unix can also be found at [30].

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 [31] and experiments in Emulab [16] 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 ms.

For experiments on real systems in Emulab [16], 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 (Fig. 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 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 Fig. 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 on the representative path*, 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

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

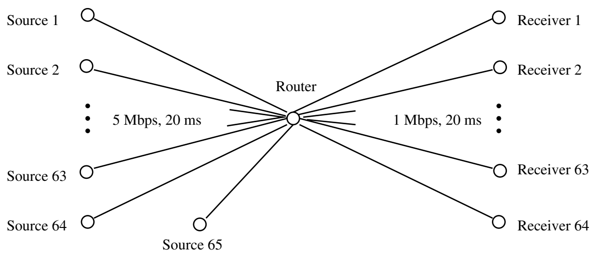

Fig. 8. 64-receiver star topology with TCP background traffic.

than 1/10 and is thus negligible. This has the same effect as measuring sequence number increment in [2]. Results in Fig. 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.

5.2. 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 Fig. 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, 0.33 Mbps. The result in Fig. 13(a) shows that ERMCC approximately achieves proportional fairness. Similar fairness achieved by PGMCC and TFMCC in the same configuration is also shown in the figure.

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 shown in Fig. 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 s. 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 Fig. 14(a) and (d). Vertical dashed

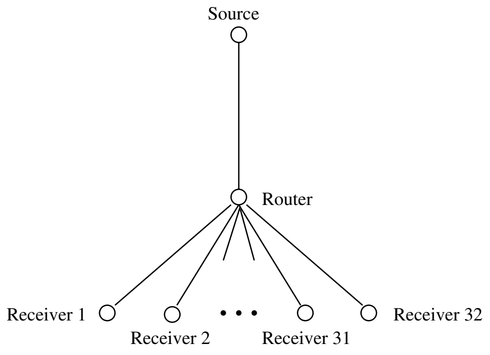

Fig. 10. One-level tree with 32 receiver nodes.

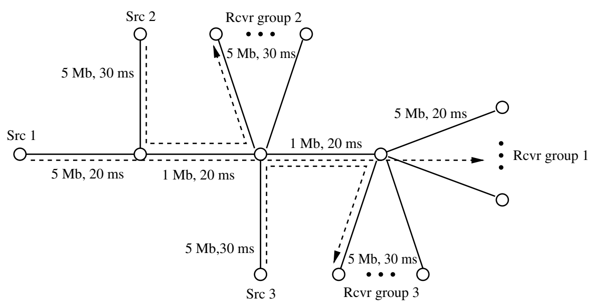

Fig. 9. Linear network with multiple bottlenecks including a total of 48 receivers.

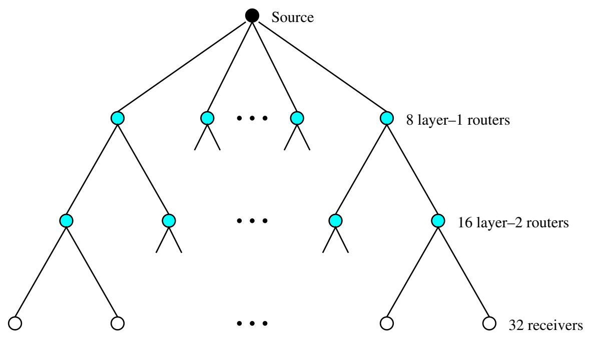

Fig. 11. Heterogeneous dynamic network.

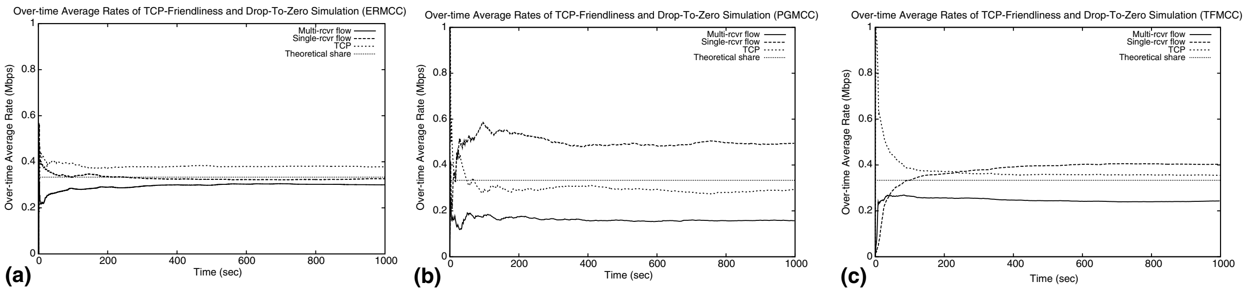

Fig. 12. TCP-friendliness and immunity to drop-to-zero: ERMCC is more TCP-friendly and avoids drop-to-zero better than PGMCC and TFMCC. (a) ERMCC, (b) PGMCC and (c) TFMCC.

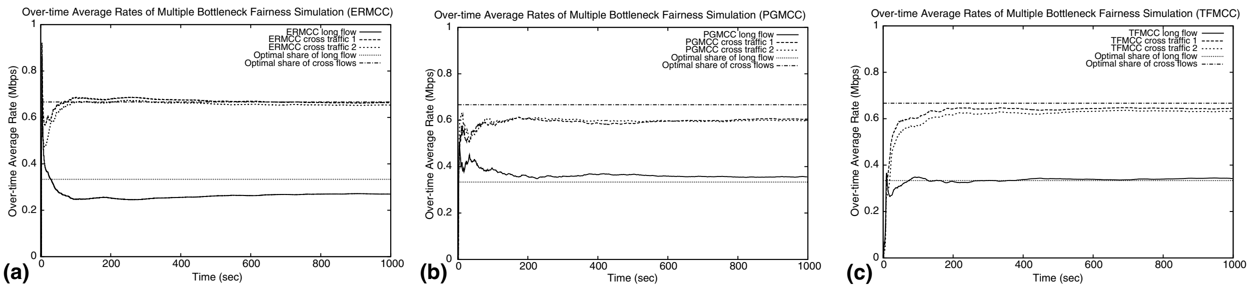

Fig. 13. Fairness of sharing bottleneck bandwidth: all three schemes achieve approximately proportional fairness. (a) ERMCC, (b) PGMCC and (c) TFMCC.

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 Fig. 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 feedbacks 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 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, $34601/(34601 + 816) \times 100\% \approx$ **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 Fig. 11, each link has 2 Mbps bandwidth. 2 links at the first level, 4 links at the second level, and 8 links at the third level has 200 ms delay.

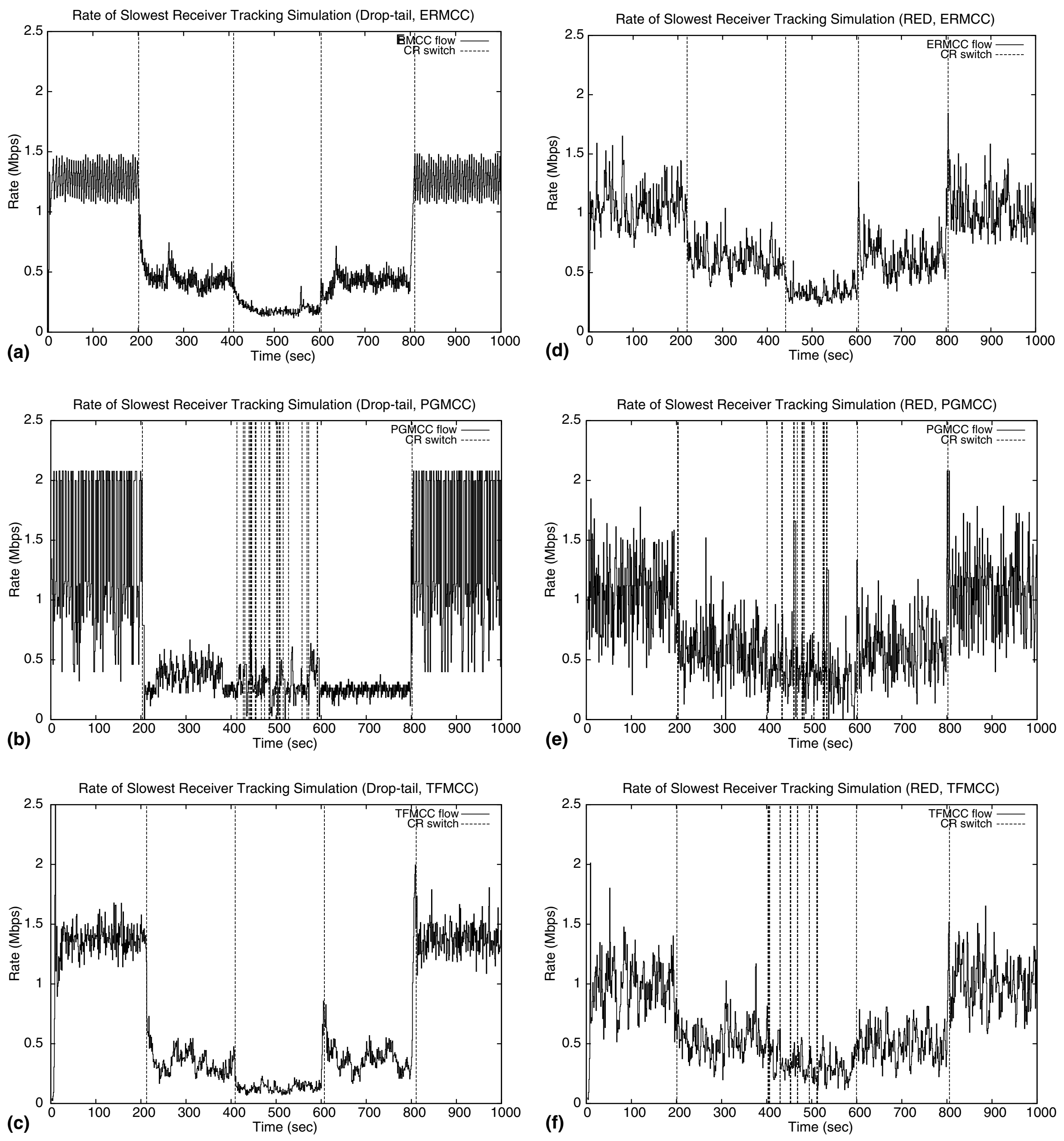

Fig. 14. Capability of tracking the slowest receiver: ERMCC tracks the slowest receiver in time with fewer representative switches. The most congested bottleneck is changed at every 200 s. For the five 200 s periods of the 1000 s simulation, the most congested bottlenecks are link 1, link 2, link 3, link 2, and link 1 in the topology shown in Fig. 10. (a) Droptail Queue (ERMCC), (b) Droptail Queue (PGMCC), (c) Droptail Queue (TFMCC), (d) RED Queue (ERMCC), (e) RED Queue (PGMCC) and (f) RED Queue (TFMCC).

All other links have 20 ms delay, while on any path between the source and a receiver, there is at most one link of 200 ms delay. On each link, two TCP Reno flows are randomly turned on and off according to Pareto distribution with average value of 60 s, and two UDP flows of 200 Kbps on and off with average value of 1 s. 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

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	126.6 Kbps	226.7 Kbps
Average number of feedbacks	**866.9**	5009.6 (NAK only)	3312.9

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 500 Kbps 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.*

We would like to note that the throughput difference between ERMCC and the other schemes mainly happens because of the methodological difference of the slowest receiver selection. ERMCC achieves better throughput because the *the slowest receiver selection* method to variations is more stable. ERMCC chooses the slowest receiver based on *Average TRAC* and the receivers apply a very effective feedback suppression, both of which causes ERMCC to respond a little slower and not to select a *transient* slowest receiver as "the slowest". This results in higher throughput in comparison to TFMCC and PGMCC.

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 [16].[6] The operating system we used is RedHat 7.1, and mrouted [32] is used for multicast routing. On the topology shown in Fig. 15, the links between the peripheral nodes and their parent nodes have 50 ms propagation delay and 1.0 Mbps bandwidth. All other links have 100 Mbps bandwidth and 0 ms propagation delay.[7] From the center node to any peripheral node, there

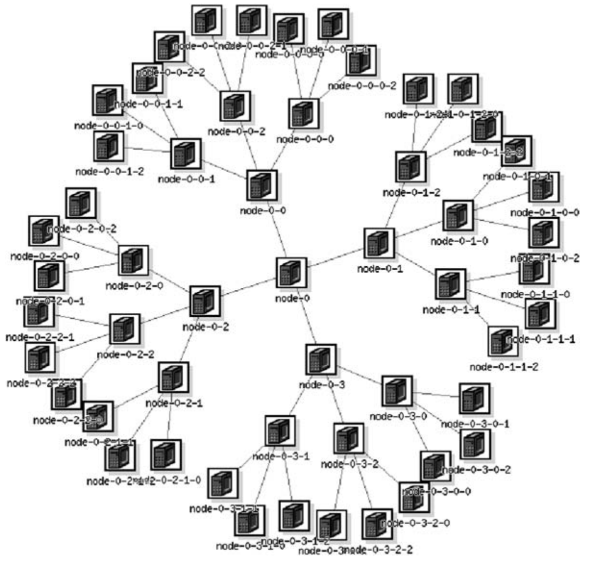
Fig. 15. Topology used in Emulab for TCP-friendliness and drop-to-zero tests with 36 receiver nodes.

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 s.

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 Fig. 16,[8] when $\beta = 0.5$, the TCP flows got more 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.

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

[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.

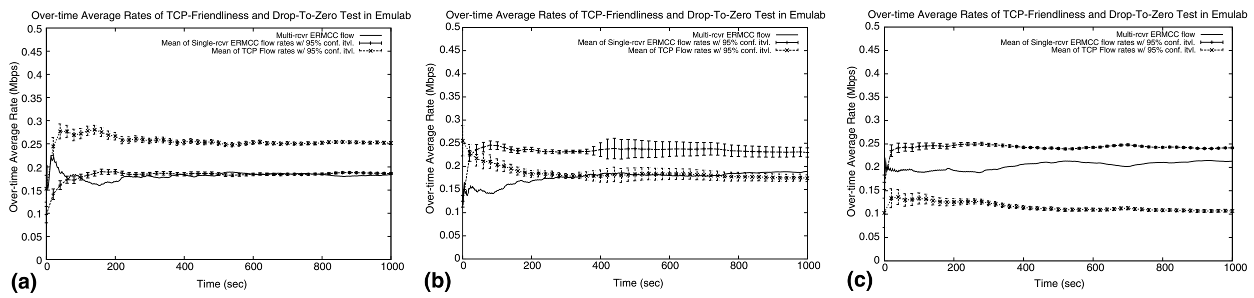

Fig. 16. TCP-friendliness and drop-to-zero test result in Emulab: ERMCC is TCP-friendly and avoids drop-to-zero on real systems with proper β setting. (a) $\beta = 0.5$, ERMCC is less aggressive than TCP; (b) $\beta = 0.65$, ERMCC is TCP-friendly; (c) $\beta = 0.75$, ERMCC is more aggressive than TCP.

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 [16]. 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 and TFMCC under most situations. Code for *ns*-2 and Unix is available at [30] for public use.

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. A related issue to investigate is the optimal length of the TRAC measurement interval Δt as it depends on the burstiness of the cross traffic and sensitivity of the application to loss.

To achieve maximal and more consistent TCP-friendliness, ERMCC would benefit from a fully automatic methodology for tuning the its rate reduction parameter, β. As we have shown in Appendix B.3, the optimal value of β is $1/2 + d/W$ where W is the bottleneck bandwidth on the representative path and d is the increase in the transmission rate of the co-existing TCP flow(s) from the time the bottleneck started to fill up to the time it drops the first packet. From this analysis, the operational range of β is $(0.5, 1)$, and our simulation and implementation based analysis of ERMCC suggests usage of β values such as 0.65 and 0.75 for different scenarios. However, an automatic tuning mechanism for β requires non-trivial estimations of d and W, which is out of scope of this paper and is a future work for investigation of ERMCC-like schemes.

Another future work item is to design application-specific (e.g., multimedia-streaming) multicast congestion control. In some cases selection of a particular receiver might affect the aggregate receiver utility since application-specific goals might not be captured by generic assumptions of single-rate multicast congestion control.

Table 3
Summary of comparative simulations

Property	Comparison w/PGMCC & TFMCC	Figure/table
TCP friendliness on the representative path	+/−	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

Acknowledgements

This work was supported in part by NSF-ITR 0313095, ARO grant W911NF-04-1-0300 and DAAD19-00-1-0559, and grants from Intel and Reuters.

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. We are also grateful to the anonymous reviewers who provided very valuable feedback about the paper.

Appendix 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 $\lambda \geqslant \hat{\mu} + 4\hat{\sigma}$ and cr_response_timer is false then
      $t_0 \leftarrow t$
      cr_response_timer ← true
    endif
  endif
  if $t - t_0 \geqslant E[T] + 8T_\sigma$ then
    cr_valid ← false
    cr_response_timer ← 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 $\lambda$ has not been cut within the most recent $\widehat{RTT}$ then
    $\lambda \leftarrow \min(\lambda, 0.75\Omega_r)$
    cr_valid ← true
    cr_response_timer ← false
  endif

Subroutine: UpdateStats ()
  Update $\hat{\mu}$ and $\hat{\sigma}$ with $\Omega_r$
  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 $T_\sigma$ 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 $\Omega_i$ and update $\mu_i$
      if $\hat{\mu}$ and $\hat{\sigma}$ are invalid or $\mu_i < \hat{\mu} - \hat{\sigma}$ then
         Send a feedback to the source
      endif
   endif
```

Appendix 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 (see Table B.1 for additional mathematical notations).

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.

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

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

[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.

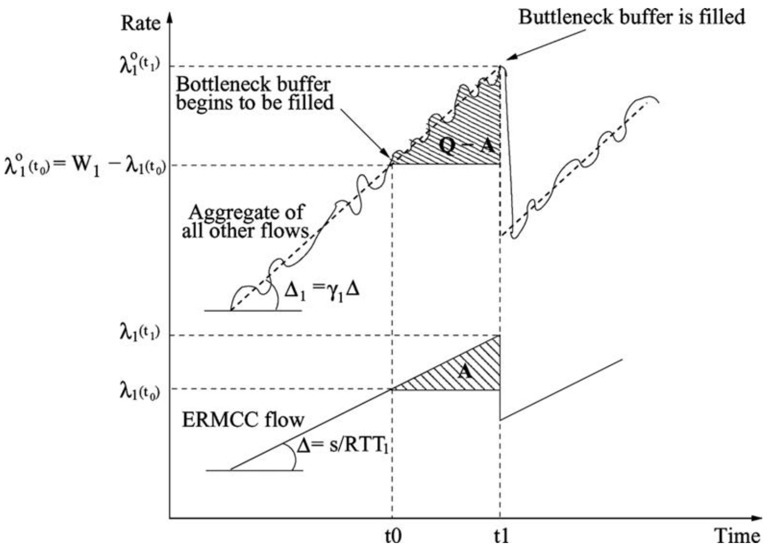

Fig. B.1. Evolution of the ERMCC flow's sending rate on the representative path.

continuously, as well as that all packet losses are due to congestion. Also, drop-tail buffer management is assumed for bottlenecks.[11]

Let us consider path 1 first (Fig. 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.,

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

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,

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

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 average 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 particularly 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 negligible, since at time t_1 the bottleneck is working at its full load, we know that the following equality holds:

[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.

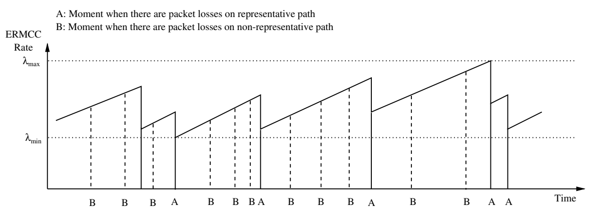

Fig. B.2. The ERMCC source only considers the congestion indications on the representative path for rate adaptation.

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

On any other path j ($j = 2 \ldots N$), since the ERMCC source ignores the congestion indications on this path (Fig. B.2), the sending rate of the ERMCC 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 $\lambda_{\min}$ and $\lambda_{\max}$ (Fig. 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 $i = 1 \ldots N$, 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 negligible, 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:

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

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

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

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 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* (Fig. B.3).

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 considering the two possible cases as we identified 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,

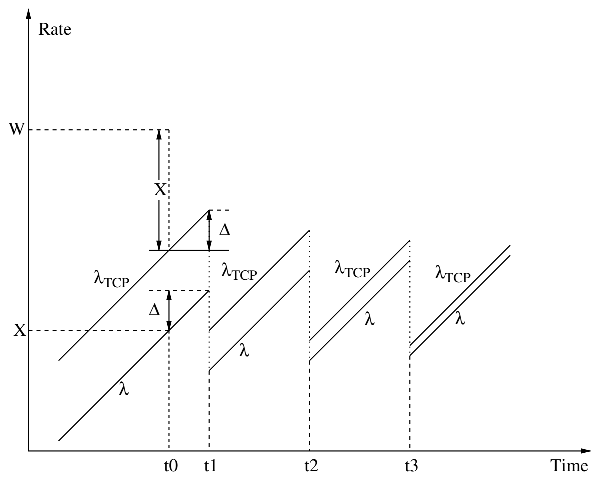

Fig. B.3. Evolution of the sending rates for TCP and ERMCC flows.

$j = 2\dots N$ is slower than R_1. This means R_j is supposed to be the new CR.

- *Case II*: R_j *worsens.* The current CR stays the same, but another flow R_j, $j = 2\dots N$ 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. Fig. 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 $\Delta_1^{\mathrm{o}} \leqslant -\Delta$.

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 Fig. 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:

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

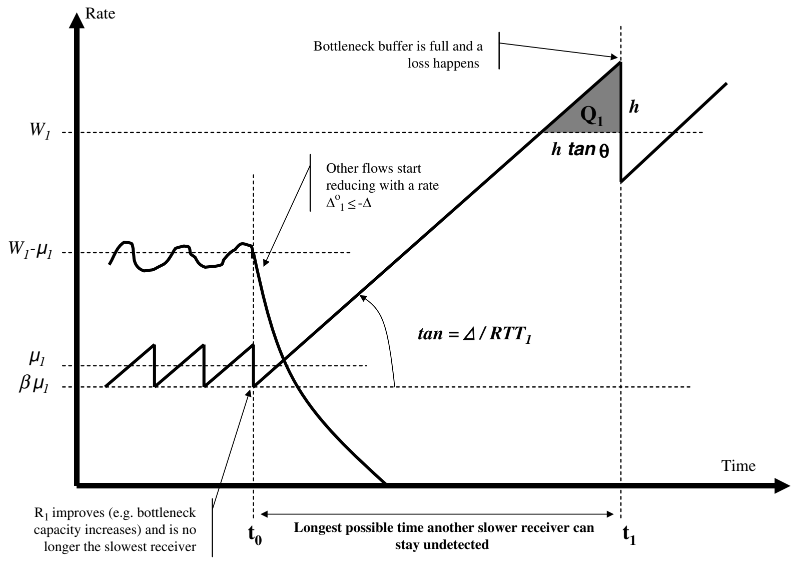
Rate
Bottleneck buffer is full and a loss happens
W_1
Q_1
h
$h \tan\theta$
Other flows start reducing with a rate $\Delta^{o}_{1} \leq -\Delta$
W_1-μ_1
$\tan = \Delta / RTT_1$
μ_1
$\beta\mu_1$
Time
R_1 improves (e.g. bottleneck capacity increases) and is no longer the slowest receiver
t_0
Longest possible time another slower receiver can stay undetected
t_1

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

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

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

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

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

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

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 $2\max(RTT_i)$. $\square$

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 Lemmas 1 and 2. $\square$

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, $\lambda(t)/\lambda^{\mathrm{TCP}}(t)$ oscillates around 1, where $\lambda(t)$ and $\lambda_{\mathrm{TCP}}(t)$ 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 Fig. B.3.

Like other TCP throughput analysis papers [17,18] 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 $\lambda_{\mathrm{TCP}}(t) + \lambda(t) \geqslant W$. Suppose some packets are lost and both flows reduce their transmission rates at t_1 (Fig. B.3). Before the losses, since both $\lambda_{\mathrm{TCP}}(t)$ and $\lambda(t)$ keep increasing, there must be a moment t_0 when $\lambda_{\mathrm{TCP}}(t_0) + \lambda(t_0) = W$. In short, let $\lambda(t_0) = X$, then $\lambda_{\mathrm{TCP}}(t_0) = W - X$. For the first step of analysis, we will show that with appropriate β,

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

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.,

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

After a reduction by half

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

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

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

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

According to the rate adaptation policy of ERMCC

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

Therefore,

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

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

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

Since $W > X$ and $d \geqslant 0$, $2/(W-X+d) > 0$, 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 (Fig. B.3). Let sending rates after reduction be $\lambda_{\mathrm{TCP}}(t_i)$ 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 $\lambda_{\mathrm{TCP}}(t_{i-1}) + \lambda(t_{i-1}) = W$. We can write the following relationship:

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

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, $\lambda(t)/\lambda_{\mathrm{TCP}}(t)$ 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 $d \geqslant 0$, 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., [33,34,3,19,35]) 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 [19].

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

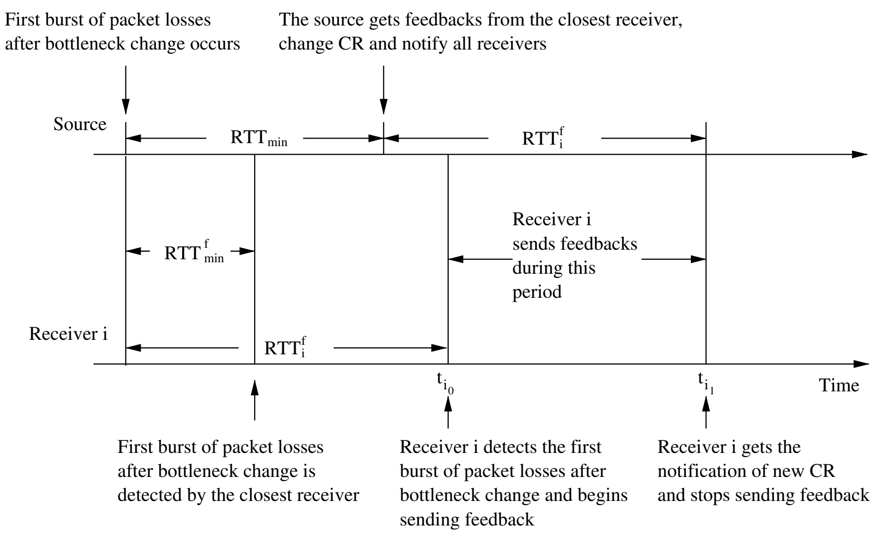
First burst of packet losses after bottleneck change occurs
The source gets feedbacks from the closest receiver, change CR and notify all receivers
Source
RTT$_{min}$
RTT$_i^f$
RTT$^f_{min}$
Receiver i sends feedbacks during this period
Receiver i
RTT$_i^f$
t_{i_0}
t_{i_1}
Time
First burst of packet losses after bottleneck change is detected by the closest receiver
Receiver i detects the first burst of packet losses after bottleneck change and begins sending feedback
Receiver i gets the notification of new CR and stops sending feedback

Fig. B.5. Feedback suppression.

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] Consider 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., $RTT_{\min} = \min_{i=1\ldots M} RTT_i$. Finally, let p be the packet loss rate experienced by the receivers behind the new MCB.[14]

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 Fig. 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 us call the period between two successive MCB switchings as *MCBSP* (*MCB Switching Period*). During a MCBSP of length $t \geqslant RTT_{\min}$, 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., $1/\delta \geqslant RTT_{\min}$. For a multicast session with average duration of T, the total number of feedbacks transmitted will approximately be[15]:

[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.

[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.

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

We can see that, for a certain T,

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

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.

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] J. Li, S. Kalyanaraman, Generalized multicast congestion control, in: Proceedings of 5th COST 264 International Workshop on Network Group Communications (NGC 2003) and ICQT, 2003.

[10] G.-I. Kwon, J. Byers, Smooth multirate multicast congestion control, in: Proceedings of IEEE INFOCOM, 2003.

[11] IETF Reliable Multicast Transport (RMT). Available from: <http://www.ietf.org/html.charters/rmt-charter.html>.

[12] J.W. Byers, M. Luby, M. Litzenmacher, A. Rege, A digital fountain approach to reliable distribution of bulk data, in: Proceedings of ACM SIGCOMM, 1998, pp. 56–57.

[13] C.K. Miller, Multicast Networking and Applications, Addison-Wesley, 1999.

[14] D. Clark, D. Tennenhouse, Architectural considerations for a new generation of protocols, in: Proceedings of ACM SIGCOMM, 2000, pp. 201–208.

[15] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, B. Vandalore, The erica switch algorithm for ABR traffic management in ATM networks, IEEE/ACM Transactions on Networking 8 (1) (2000) 87–98.

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

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

[18] M. Mathis, J. Semke, J. Mahdavi, T. Ott, The macroscopic behavior of the tcp congestion avoidance algorithm, ACM Computer Communications Review 27 (3) (1997).

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

[20] S. McCanne, V. Jacobson, M. Vetterli, Receiver-driven layered multicast, in: Proceedings of ACM SIGCOMM, 1996.

[21] L. Vicisano, L. Rizzo, J. Crowcroft, Tcp-like congestion control for layered multicast data transfer, in: Proceedings of IEEE INFOCOM, 1998.

[22] A. Legout, E. Biersack, Plm: fast convergence for cumulative layered multicast transmission schemes, in: Proceedings of ACM SIGMETRICS, 2000.

[23] I.E. Khayat, G. Leduc, Congestion control for layered multicast transmission, Networking and Information Systems Journal 33 (4) (2000) 559–573.

[24] M. Kawada, H. Morikawa, T. Aoyama, Cooperative inter-stream rate control scheme for layered multicast, in: Proceedings of Symposium on Applications and the Internet (SAINT 2001), 2001, pp. 147–154.

[25] J.W. Byers et al., Flid-dl congestion control for layered multicast, in: Proceedings of NGC, 2000.

[26] D. Sisalem, A. Wolisz, Mlda: a tcp-friendly congestion control framework for heterogeneous multicast environments, in: Proceedings of IWQoS, 2000.

[27] J. Liu, B. Li, Y.-Q. Zhang, A hybrid adaptation protocol for tcp-friendly layered multicast and its optimal rate allocation, in: Proceedings of IEEE INFOCOM, 2002.

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

[29] T. Nguyen, K. Nakauchi, M. Kawada, H. Morikawa, T. Aoyama, Rendezvous points based layered multicast, IEICE Transactions on Communication E84-B (12) (2001).

[30] Ermcc source code. Available from: <http://networks.ecse.rpi.edu/source_code.html>.

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

[32] 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>.

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

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

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

Jiang Li is an Assistant Professor in the Department of Systems and Computer Science at Howard University, Washington, DC, USA. He received his B.S. and M.S. degree in Computer Science from the University of Science and Technology of China, Hefei, China in 1995 and 1998 respectively. In August 2003, he graduated from Rensselaer Polytechnic Institute in Troy, NY, USA with a Ph.D. degree in Computer Science. His research areas are in computer networking. In particular, he is interested in flow and congestion control, IP and application layer multicast, wireless networks, sensor networks, peer-to-peer networks, overlay networks and network security. He is currently conducting research on delay tolerant networks and multicast congestion control. He is a member of IEEE and ACM.

Murat Yuksel is a Post-Doctoral Researcher and Lecturer at ECSE Department of Rensselaer Polytechnic Institute, Troy, NY. He received M.S. and Ph.D. degrees in Computer Science from Rensselaer Polytechnic Institute in 1999 and 2002 respectively. He holds a B.S. degree in Computer Engineering from Ege University, Izmir, Turkey in 1996. His research is on various networking issues such as routing in wireless sensor and adhoc networks, mobile free-space-optical networks, large-scale network simulation, network pricing, and performance analysis. He is a member of IEEE and Sigma Xi.

Shivkumar Kalyanaraman is an Associate Professor at the Department of Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute in Troy, NY. He received a B.Tech degree from the Indian Institute of Technology, Madras, India in July 1993, followed by M.S. and Ph.D. degrees in Computer and Information Sciences at the Ohio State University in 1994 and 1997, respectively. His research interests are in network traffic management topics such as congestion control architectures, quality of service (QoS), high-speed wireless, free-space optical networking, network management, multicast, pricing, multimedia networking, and performance analysis. His special interest lies in developing the inter-disciplinary areas between traffic management, wireless communication, optoelectronics, control theory, economics, scalable simulation technologies, and video compression. He is an member of the ACM and IEEE.