ORMCC : A Simple And Effective Single-Rate Multicast Congestion Control Scheme

Jiang Li, Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180

Abstract— **Owing to the simplicity and ease of deployment, single-rate multicast congestion control is worthy of further exploration. In this article, we propose a new single-rate multicast congestion control scheme called *ORMCC* based on a new metric, *TRAC (Throughput Rate At Congestion)*. The scheme is simple: states maintained at source and receivers are $O(1)$; only simple computation is required; and there is no need to measure RTTs from all receivers to the source. At the same time, the scheme is TCP-friendly, does not suffer from *drop-to-zero* problem, and is very effective with feedback suppression. Theoretical analysis of the scheme performance is provided, and simulations have shown that ORMCC outperforms PGMCC [15] and TFMCC [20] under most situations. We have also implemented ORMCC on top of UDP and successfully run it on real systems in Emulab[19] with promising results.**

Index Terms— **Multicast, congestion control, single-rate, drop-to-zero, TCP friendliness, feedback suppression, throughput rate at congestion (TRAC)**

I. 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 [4], PGMCC[15], TFMCC[20], MDP-CC[9] and our prior work LE-SBCC [18]. 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 [2] and STAIR [3]. The single-rate category is easy to implement and deploy, because it does not require support from intermediate nodes beyond standard multicast capabilities, also does not introduce high processing load to them. Although such schemes do not scale as well as multi-rate ones because they track the slowest receiver, they are suitable for such situations as the multicast in a not-so-heterogeneous environment, or bulk data transfer without concerns over delay. With some network support [8], we can also emulate multi-rate schemes by deploying single-rate schemes on selected intermediate nodes.

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

The contributions of this paper include:

- It proposes ORMCC, a simple and effective single-rate multicast congestion control scheme based on a new metric TRAC, with features such as $O(1)$ state, *non-timer-based* feedback suppression etc.
- It analyzes the scheme performance theoretically.
- By comparison in simulation with PGMCC [15] and TFMCC [20], it shows that ORMCC achieves better performance under most situations.
- It includes the test results of ORMCC implementation on real systems in Emulab [19].

Our scheme is simple because (1) at the source and receivers, $O(1)$ state is maintained, and only simple computation is required, (2) 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 (3) 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 problems of slowest receiver tracking, TCP-friendliness, and drop-to-zero, (2) the feedback suppression mechanism works very effectively by suppressing over 95% feedback under normal situations. In fact, it outperforms PGMCC[15] and TFMCC[20] under most situations. The key idea of ORMCC 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.

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. When there is no CR, all receivers may send feedbacks to the source. 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 appli-

cable to both reliable and unreliable multicast.

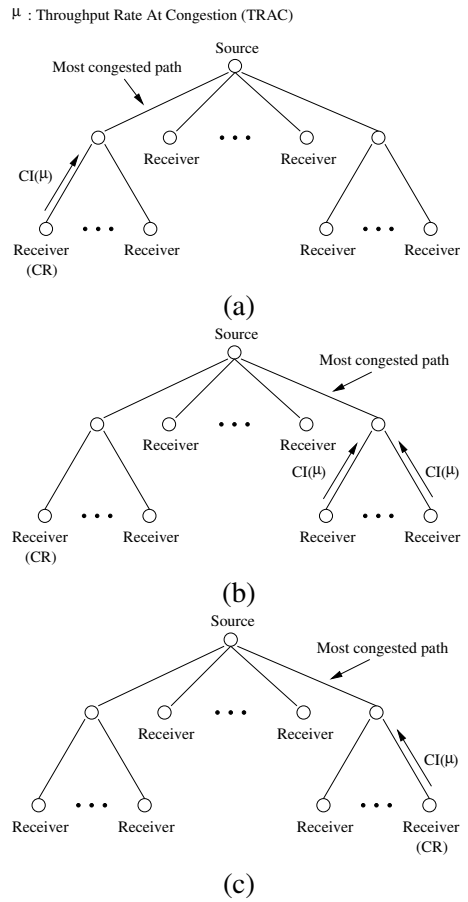


Fig. 1. EXAMPLE OF ORMCC OPERATION

An example operation can illustrate how our scheme works more clearly. In Figure 1 (a), 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. The feedback is CI(μ) in Figure 1 where CI means *congestion indication* and μ is TRAC measured by receiver. After a while, another path becomes the new most congested path. Those receivers behind that path will see average TRACs lower than that of the current CR, and will send feedbacks (Figure 1 (b)). As a result, one of them will be chosen as the new CR. After that, again, other receivers suppress their feedback (Figure 1 (c)).

In the following sections, we will briefly discuss some related work, followed by the ORMCC details. Simulation and experiment results are presented at the end.

II. Related Work

A. Single-Rate Schemes

DeLucia et. al's work in [4] 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 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"[4], 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 [15], TFMCC [20] and MDP-CC [9] are recent work also using representatives. Although they use different policies for rate adaptation, they all leverage the TCP throughput formula [14] [10] 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 [15] 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 [20] adjusts the rate according to the estimated rate calculated by the representative. RTTs are measured by receivers with a somewhat complex procedure. The sender needs to echo receiver's feedback according to some priority order, and there is one-way delay RTT adjustment plus sender-sider RTT measurement. TFMCC comes with feedback suppression which is an enhanced

version of [7] 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 [9] 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 [18] 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, it needs $O(n)$ (n is the number of receivers) states at the source, and network aggregation can also lead to performance degradation. ORMCC does not have these drawbacks.

B. Multi-Rate Schemes

Ideally, the multi-rate multicast congestion control can satisfy heterogeneous receivers because each of them receives data at its own rate. The most noticeable among them are recently developed Fine-Grained Layered Multicast [2] and STAIR [3]. However, the multi-rate schemes are *closely coupled* with routing and IGMP, which implies some potential problems. For example, different groups for layers could follow different routes [12]. Aggregated multicast trees [5] do not necessarily prune trees dynamically and hence break the assumptions of the multi-rate schemes. The slackness of response to congestion due to long leave latency continues to be an issue. Besides, frequent group joins and leaves can introduce significant load at routers.

III. ORMCC Details

As we have mentioned in the introduction, in ORMCC, receivers send 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, followed by a list of the features of ORMCC.

A. Details of ORMCC Operations

1) ***Feedback Required from Receivers – CI(μ):*** When receivers detect congestion by packet losses, they need to inform the source so that the source can adjust the transmission rate accordingly. In ORMCC, the feedback is *Congestion Indications with TRAC (CI(μ)s)*. Like NAKs, CI(μ)s may be sent only when a receiver detects packet losses (though they may also be suppressed). Assume that at the arrival of packet A, a receiver detects that some packets have been lost. It will then send a CI(μ) to the source. Such a CI(μ) contains (1) The sequence number of A, for the sake of RTT measurement, and (2) The output rate measured at the arrival of A, for the sake of CR allocation. Note than when multiple packet losses are detected by the arrival of one packet, only one CI(μ) will be sent (if without suppression), while multiple NAKs are generated for the same situation. With suppression, CI(μ)s may be sent at a less frequency.

To avoid oscillation, we average the output rate over a short period of time.[1] Note that the output rates used here are different from those in normal sense, because they are measured only when packet losses are detected due to congestion. To distinguish them, we give the notation *Throughput Rate At Congestion (TRAC)*.

2) ***Allocation of The Slowest Receiver:*** ORMCC compares average TRAC of all receivers to locate the slowest ones, and choose one of them as the *Congestion Representative (CR)*. By using TRAC, it avoids computing TCP throughput formula [10] [14] which requires RTT and packet loss rate.

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 CI(μ)s from that receiver for rate adaptation. Average TRAC is calculated by means of *EWMA (Exponentially Weighted Moving Average)*. Denote TRAC as U, average TRAC as $E(U)$. With a sample U, $E(U)$ is updated as $E(U) \leftarrow (1-\alpha)E(U) + \alpha U$. Deviation U_σ is also updated as $U_\sigma \leftarrow (1-\alpha)U_\sigma + \alpha|E(U) - U|$ for the sake of CR updating and feedback suppression (see Section III-A.3 and III-A.4).

The receivers help the source to select a receiver with the lowest average TRAC by sending in CI(μ)s only if their average TRACs are low enough to qualify them as CR. More details of how receivers help will be covered in Section III-A.4 of feedback suppression.

3) ***Update of CR under Dynamic Conditions:*** 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: (1) The situations of some non-CR receivers

[1] In our simulations and implementation, we use one second.

change so that one of them sees more severe congestion than the current CR does. (2) 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 situation (1), but to cope with situation (2) needs more effort. Under this situation, there can be no CI(μ)s from the current CR. Recall that the source only consider the CI(μ)s from the CR for rate adaptation and ignores all other CI(μ)s. If the source does not change CR in time, the transmission rate will be out of control. To detect that, we estimate an upper bound (denoted as T^{cr}_{max}) of the idle time (denoted as T^{cr}) before the source receives a first CI(μ) from the CR when the bottleneck is fully loaded. To use T^{cr}_{max}, suppose we somehow detect that the bottleneck is fully loaded at time t. If there has been no CI(μ) from the current CR for T^{cr}_{max} since t, we can say that the current CR is now inactive and needs to be changed.

Let's look at Figure 2. When the CR is still active, we measure samples of T^{cr} at the source, using feedback packets only from CR. When the transmission rate reaches $E(U^{cr}) + 4U^{cr}_{\sigma}$[2], where $E(U^{cr})$ and U^{cr}_{σ} are the average and deviation of the current CR's TRAC respectively, we assume that bottleneck becomes fully loaded and start to count. Let the current time be t_0. At a later time t_1, the first CI(μ) since t_0 arrives at the source from the CR. $t_1 - t_0$ is then a sample of T^{cr} and we update the average and deviation of T^{cr} with EWMA. T^{cr}_{max} is the average value of T^{cr} plus eight times its deviation[3]. When the CR is not active, for the duration of T^{cr}_{max} since we start to count, no CI(μ) will be received by the source. The source then requests feedback from other receivers for new CR selection, as described in Section III-A.4.

There is one small trick we use to bias the choice of CR toward those receivers with higher RTTs. After a new CR is chosen, we set a grace period of $2RTT_{max}$, where RTT_{max} is the maximum RTT the source has ever seen. Within this period, if the source receives a CI 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. Grace period is not reset after CR switches within grace period.

*4) **Feedback Suppression by Receivers:*** Effective feedback suppression can reduce the risk of feedback implosion, and allow a multicast congestion control scheme to be used for large groups. In ORMCC, the source conveys the average $E(U^{cr})$ and the deviation U^{cr}_{σ} of the CR's TRAC to receivers whenever the CR is updated or $E(U^{cr})$ and U^{cr}_{σ} are changed. Only if a receiver's own average TRAC $E(U)$ is less than $E(U^{cr}) - U^{cr}_{\sigma}$[4], will it send CI($\mu$)s. Note that a receiver does not maintain TRAC deviation U_{σ} because it is computed by the source with the help of TRAC in CI(μ)s.

$E(U^{cr})$ and U^{cr}_{σ} conveyed by the source can be set to infinitely large values so that all receivers can send CI(μ)s. This is needed when the current CR is inactive and the source needs to trigger feedbacks from all receivers for new CR selection (Figure 2).

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, CI(μ)s are not scheduled at all before being suppressed. Yet, it is effective since the amount of CI(μ)s sent to the source is independent of the total receiver number. More insight will be given in the theoretical analysis at Appendix II-D.

There is one situation which might concern people. When the current CR is absent and the source needs to choose a new CR, all receivers seeing congestion of similar degree may send CIs 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 CIs for CR re-selection. Therefore, the total number of CIs sent under this situation is limited, and we do not deem it as a problem.

*5) **Rate Adaptation:*** ORMCC is a rate-based scheme, using the policy of additive increase and multiplicative decrease (AIMD). That is, if there are no CI(μ)s 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 CI is received from the CR, let the TRAC in this CI be μ, we adjust the transmission rate to the minimum of $\beta\mu$ and the current rate. CI(μ)s from other non-CR receivers will be ignored, and at most one rate cut is allowed per RTT.

The rate reduction factor β is an important parameter of ORMCC. The larger the β, the more aggressive is ORMCC. To keep ORMCC TCP-friendly, from a later discussion in Appendix II-B, we will see that β must be at least 0.5. Moreover, the exact value of β depends on how ORMCC is implemented. According to the simulation and experiment results, we suggest $\beta = 0.65$ for imple-

[2]According to Chebychev Inequality, about 94% of the random sample are less than this value.

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

[4]We don't use $E(U) \geq E(U^{cr})$ because we want to be conservative and keep CR stable.

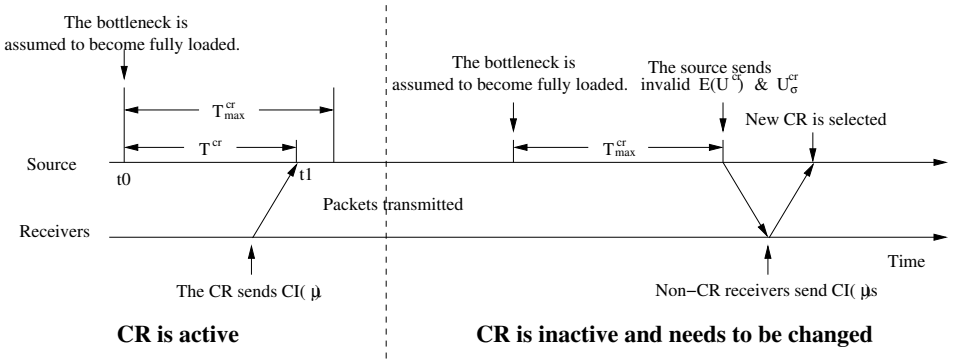


Fig. 2. UPDATE OF CONGESTION REPRESENTATIVE (CR)

mentation on user level, and $\beta = 0.75$ for implementation in system kernel. The reason is that, if ORMCC 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.

*6) **RTT Estimation:*** Unlike a NAK, which includes the sequence number of a lost packet, a CI(μ) of ORMCC 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 CI(μ) 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 CI(μ)s are sent only when packet losses occur, RTT estimated by CI(μ)s includes the maximum bottleneck queueing delay and thus is still the upper bound. On the other hand, ACKs as those in TCP may or may not include bottleneck queueing delay. Therefore, on average, RTT estimated by CI(μ)s 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.

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

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

The pseudo code of ORMCC's algorithm is provided in Appendix I for reference. The code for ns-2 and Unix can also be found at [21].

IV. PROPERTIES ABOUT ORMCC PERFORMANCE

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

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

Proof: (See Appendix II-A.)

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

Proof: (See Appendix II-B.)

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

Proof: (See Appendix II-C.)

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

Proof: (See Appendix II-D.)

V. SIMULATIONS AND EXPERIMENTS

We have run simulations on ns-2 [1] and experiments in Emulab [19] to validate the performance of ORMCC. 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[15] and TFMCC[20] and compared the performance of our scheme with theirs. For ORMCC and TFMCC, we use ns2.1b7a, for PGMCC, we use ns2.1b5, due to the restriction of its source code. In all simulations,

the data packet size is 1000 bytes, the bottleneck buffer size is 50K bytes, the initial RTT is 100 milliseconds.

For experiments on real systems in Emulab[19], we implemented ORMCC 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.

A. TCP-Friendliness and Drop-To-Zero Avoidance

We used a star topology (Figure 3) 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 ORMCC flow. Furthermore, there is a multi-receiver ORMCC flow from source 65 to all 64 receivers. Therefore, on a path between the router and any receiver, the multi-receiver ORMCC flow competes with a TCP flow and a single-receiver ORMCC flow.

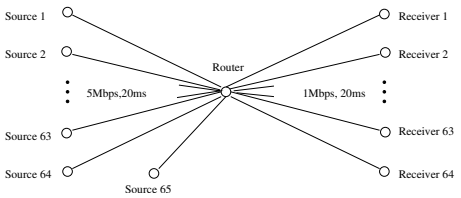


Fig. 3. 64-RECEIVER STAR TOPOLOGY WITH TCP BACKGROUND TRAFFIC

We randomly chose a receiver node and plot in Figure 4(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 ORMCC flow and the multi-receiver ORMCC flow are close to each other indicates that *(1) ORMCC is TCP-friendly, and (2) ORMCC does not suffer from drop-to-zero problem.*

We also conducted experiments on the same configuration for PGMCC[6] and TFMCC. Results in Figure 4 (b) and (c) show that but the average rates of their multicast flows deviate more from corresponding unicast flows.

B. Multiple Bottleneck Fairness

In real world, there are usually more than one bottleneck on a path. It is desirable to check how long ORMCC flows compete with short ones and what kind of fairness ORMCC can achieve. we ran a simulation on Figure 5. 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.

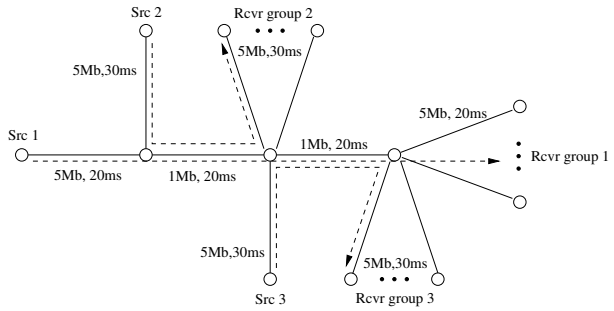


Fig. 5. LINEAR NETWORK WITH MULTIPLE BOTTLENECKS (TOTALLY 48 RECEIVERS)

According to proportional fairness, the long ORMCC flow should get one-third of the bottleneck bandwidth, 0.33Mbps. The result in Figure 6 (a) shows that ORMCC achieves approximately proportional fairness. Similar fairness achieved by PGMCC and TFMCC in the same configuration is also shown in the figure.

C. Slowest Receiver Tracking

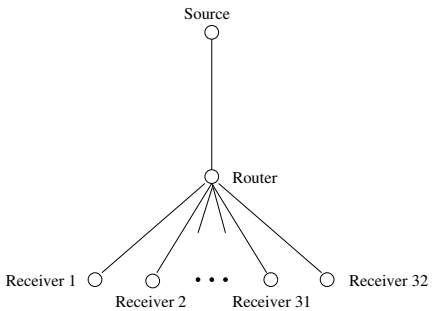
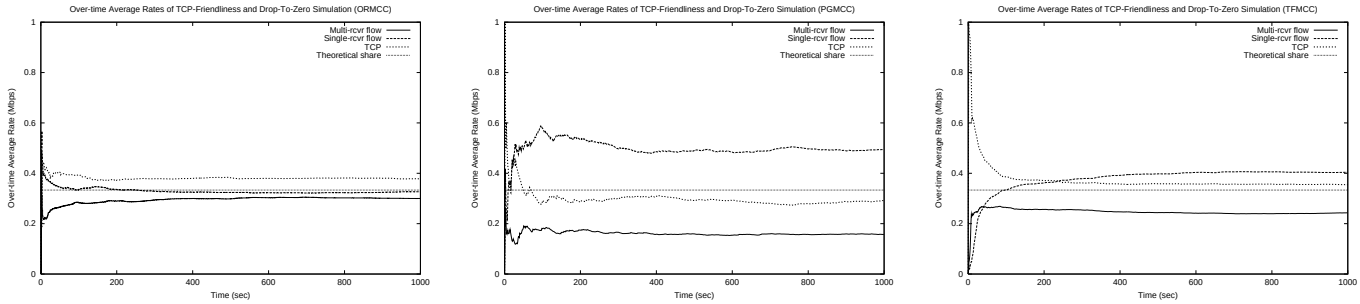


Fig. 7. ONE-LEVEL TREE WITH 32 RECEIVER NODES

This simulation is used to test ORMCC's capability to quickly track the slowest receiver and select it as CR. In the tree topology of Figure 7, there is an ORMCC flow between the source and all the 32 receivers. There are three dynamically generated bottlenecks using TCP Reno flows. Denote link i as the link between the router and receiver i ($i = 1 \ldots 32$), each link has 2Mb bandwidth and 20 ms delay. The simulation time is 1000 seconds. During the whole simulation, one TCP flow runs on link 1; between 200th and 800th seconds, three TCP flows run on link 2; between 400th and 600th seconds, seven TCP flows run on link 3. The most congested bottlenecks and the supposed CRs at different time are shown in Table I.

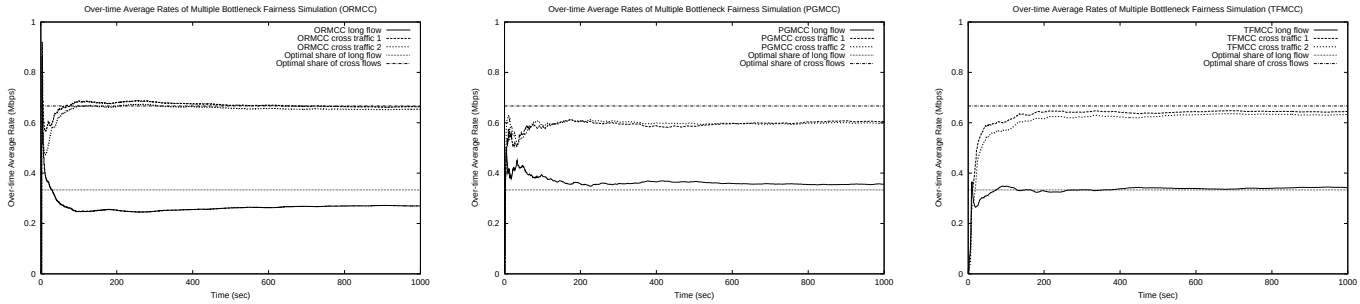
[5] Over-time average rate at time t is defined as the traffic volume between $[0, t]$ divided by t.

[6] For PGMCC simulations, since ns2.1b5 can only accommodate up to 128 end nodes, we can only have 63 pairs of unicast source and receiver instead of 64 pairs. Moreover, we only measure the sending rate of original data packets, because repair packets for PGMCC are routed by net elements to individual receivers whoever need them instead of all receivers. Nevertheless, the proportion of repair packets is less than 1/10 and is thus negligible. This has the same effect as measuring sequence number increment in [15].



(a) ORMCC (b) PGMCC (c) TFMCC

Fig. 4. TCP-FRIENDLINESS AND IMMUNITY TO DROP-TO-ZERO (ORMCC is more TCP-friendly and avoids drop-to-zero better than PGMCC and TFMCC.)



(a) ORMCC (b) PGMCC (c) TFMCC

Fig. 6. FAIRNESS OF SHARING BOTTLENECK BANDWIDTH (All three schemes achieve approximately proportional fairness.)

TABLE I
DYNAMICS IN SLOWEST RECEIVER TRACKING SIMULATION

	0 - 200 sec	200 - 400 sec	400 - 600 sec	600 - 800 sec	800 - 1000 sec
Most congested bottleneck	Link 1	Link 2	Link 3	Link 2	Link 1
Supposed CR	Receiver 1	Receiver 2	Receiver 3	Receiver 2	Receiver 1

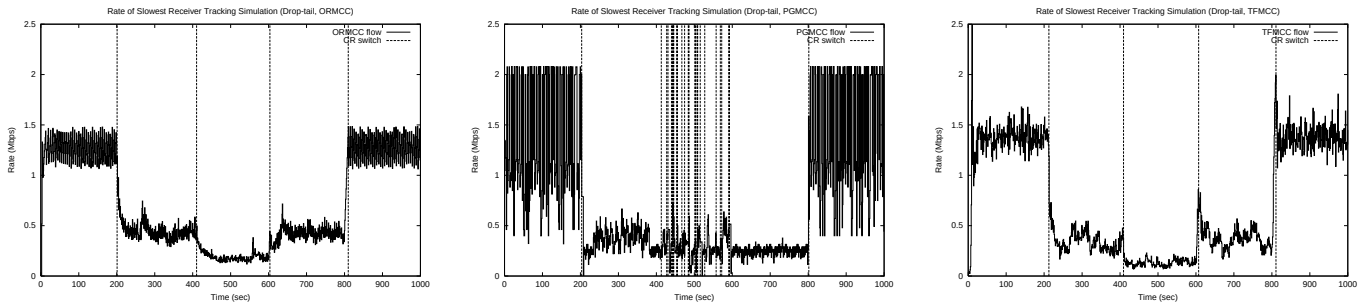
The dynamics include both conditions causing CR switches, i.e. (1) A slower receiver appears, (2) The current slowest receiver is absent. RED and drop-tail queue management policies are used separately in our simulations. Simulation results are shown in Figure 8 (a) and (d). Vertical dash lines show when the ORMCC source switched CR. We can see that ORMCC updates CR and adapts its transmission rate in a timely manner. Note that when using RED queues, the ORMCC source sometimes switched CR a little slower than drop-tail situation. The reason is because RED queue drops packets in a random manner, it takes longer for the slowest receiver to have a lower average TRAC measurement.

Under the same situation, as shown in Figure 8 (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 ORMCC and TFMCC have similar smoothness.

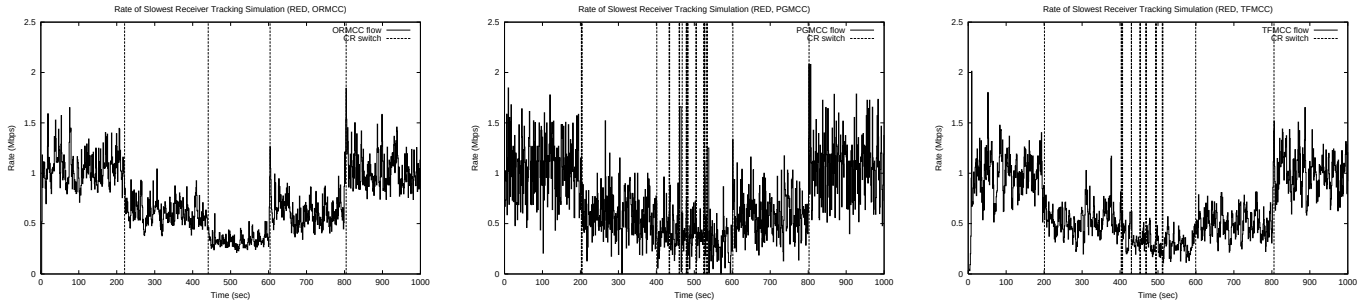
D. Feedback Suppression

To check the effectiveness of the feedback suppression mechanism in ORMCC, we refer back to the simulation of TCP-friendliness and drop-to-zero avoidance. In totally ten simulations, the average total number of CI(μ)s sent by all receivers is 816 (standard deviation is 14.8), the average total number of suppressed CI(μ)s is 34601 (standard deviation is 422.0). The average number of CI(μ)s would have been sent by a receiver if without suppression, is $(34601 + 816)/64 \approx 553$. As we discussed in the analysis (Appendix II-D), realistic measurement error can lead to a little bit more CIs. 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 CI(μ)s suppressed, $34601/(34601 + 816) \times 100\% \approx \mathbf{97.7}\%$, shows that *our feedback suppression is very effective.*

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



(a) Droptail Queue (ORMCC) (b) Droptail Queue (PGMCC) (c) Droptail Queue (TFMCC)



(d) RED Queue (ORMCC) (e) RED Queue (PGMCC) (f) RED Queue (TFMCC)

Fig. 8. CAPABILITY OF TRACKING THE SLOWEST RECEIVER (ORMCC tracks the slowest receiver in time with fewer representative switches.)

ume is much larger.

E. Comparision with PGMCC and TFMCC in Heterogeneous Dynamic Network

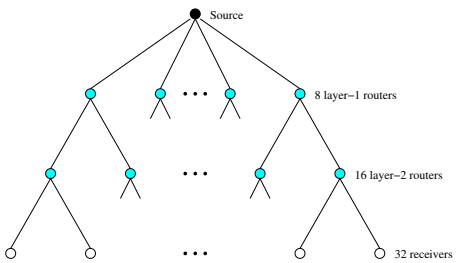


Fig. 9. HETEROGENEOUS DYNAMIC NETWORK

As the last simulation, we constructed a dynamic network to test the stability and adaptability of ORMCC, again compared with PGMCC and TFMCC. In Figure 9, each link has 2Mbps bandwidth. 2 links at the first level, 4 links at the second level, and 8 links at the third level has 200ms delay. All other links have 20ms delay, while on any path between the source and a receiver, there is at most one link of 200ms delay. On each link, two TCP Reno flows are randomly turned on and off according to Pareto distribution with average value of 60 seconds, and two UDP flows of 200Kbps on and off with average value of 1 second. These flows dynamically generate bottlenecks and make the network heterogeneous. At last, there is a multicast flow from the source to all the receivers. The multicast flow can use either ORMCC, PGMCC or TFMCC. Therefore, at any moment, there are at most five flows on any link: one multicast flow, two TCP flows and two UDP flows, and the multicast flow is expected to get an average throughput rate of 500Kbps or so.

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

F. TCP-Friendliness and Drop-To-Zero Avoidance Test in Emulab

To do a preliminary check of ORMCC's performance in real world and understand the issues of implementation, we implemented ORMCC in C++ on top of UDP as a user level program and ran it in Emulab [19] [7]. The operating system we used is RedHat 7.1, and mrouted[11] is used for multicast routing. On the topology shown in Figure 10, the links between the peripheral nodes and their parent nodes have 50 ms propagation delay and 1.0Mbps bandwidth. All other links have 100Mbps bandwidth and 0 ms propagation delay.[8] From the center node to any peripheral node, there is a single-receiver ORMCC flow and a TCP flow. Also, there is a multi-receiver ORMCC flow from the center node to all 36 peripheral nodes. The experiment time is 1000 seconds.

[7] Emulab is accessible at http://www.emulab.net.

[8] The propagation delay here means the delay artificially introduced by some particular software.

TABLE II

COMPARISON OF AVERAGE THROUGHPUT AND FEEDBACK VOLUME IN HETEROGENEOUS DYNAMIC NETWORK (ORMCC HAS HIGHER THROUGHPUT AND LESS FEEDBACK.)

	ORMCC	PGMCC	TFMCC
Average Throughput	**415.4** Kbps	126.6Kbps	226.7Kbps
Average Feedback Number	**866.9**	5009.6 (NAK only)	3312.9

Since we implemented ORMCC on user level, its traffic is more bursty than that of TCP running in kernel. As described in Section III-A.5, the rate cut factor (β) should be adjusted accordingly. We tried three different values: 0.5, 0.65 and 0.75. According to Figure 11 [9], when $\beta = 0.5$, the TCP flows got more bandwidth; when $\beta = 0.75$, the ORMCC flows are more aggressive. $\beta = 0.65$ works the best, where the multi-receiver ORMCC 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 ORMCC flow and single-receiver ones are always close, showing that ORMCC is immune to drop-to-zero problem.

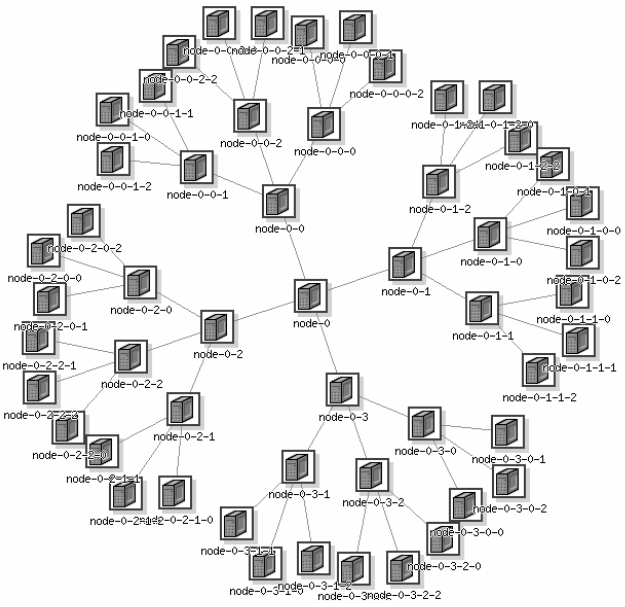


Fig. 10. TOPOLOGY USED IN EMULAB FOR TCP-FRIENDLINESS AND DROP-TO-ZERO TEST (36 receiver nodes)

VI. CONCLUSION

We have proposed a single-rate multicast congestion control scheme in this paper. It uses an conventional concept of representative named *Congestion Representative (CR)* here. However, by leveraging a new metric *TRAC*, the whole scheme is simple while still capable of effectively addressing the problems of TCP-friendliness, drop-to-zero, slowest receiver tracking and feedback suppression. The states maintained by source and receivers are $O(1)$; operations of source and receivers are all simple without requiring intense computation, in particular there is no need to measure RTTs between all receivers and the source; non-probabilistic-timer-based feedback suppression is highly effective. To confirm the performance of ORMCC, we have not only provided theoretical analysis, but also run simulations to compare our scheme with PGMCC[15] and TFMCC[20]. Furthermore, we have implemented ORMCC on top of UDP and run it on real systems in EMulab[19]. The results are promising. As an emphasis, we summarize the comparison with PGMCC and TFMCC in simulations in Table III. We can see that ORMCC achieves better performance than PGMCC and TFMCC under most situations. Code for ns-2 and Unix is available at [21] for public test.

[9]The mean and confidence interval are calculated out of all the flows of the same category.

VII. ACKNOWLEDGMENT

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

REFERENCES

[1] S. Bajaj, et al, "Improving Simulation for Network Research", *Technical Report 99-702b, University of Southern California, March 1999, revised September 1999*

[2] J. Byers, M. Luby, M. Mitzenmacher, "Fine-Grained Layered Multicast", *Infocom 2001*

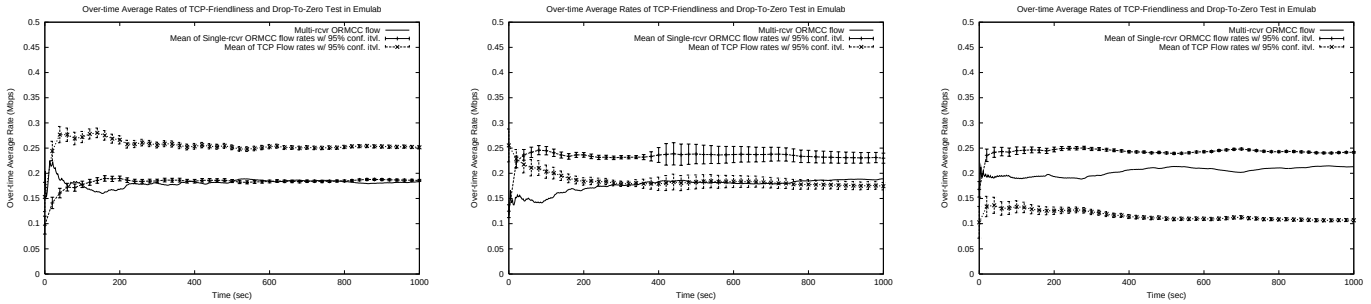
[3] J. Byers, G. Kwon, "STAIR: Practical AIMD Multirate Multicast Congestion Control", *NGC 2001*

[4] Dante DeLucia, Katia Obraczka, "A Multicast Congestion Control Mechanism Using Representatives", *Proceedings of the IEEE ISCC 1998*

[5] A. Fei, J. Cui, M. Gerla, M. Faloutsos, "Aggregated Multicast: an Approach to Reduce Multicast State", *Globecom 2001*

[6] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, Lixia Zhang, "A Reliable Multicast Framework for Light-wight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking*, Vol. 5(6): 784-803, Dec. 1997.

[7] Thomas T. Fuhrmann, Jorg Widmer, "On the Scaling of Feedback Algorithms for Very Large Multicast Groups", *Computer Communications*, 24(5-6): 539-547, Mar. 2001.



(a) $\beta = 0.5$ ORMCC is less aggressive than TCP

(b) $\beta = 0.65$ ORMCC is TCP-friendly

(c) $\beta = 0.75$ ORMCC is more aggressive than TCP

Fig. 11. TCP-FRIENDLINESS AND DROP-TO-ZERO TEST RESULT IN EMULAB (ORMCC is TCP-friendly and avoids drop-to-zero on real systems with proper β setting.)

TABLE III
SUMMARY OF PERFORMANCE COMPARISON WITH PGMCC AND TFMCC IN SIMULATIONS

Simulation Name	*Performance Comparison*	*Figure/Table*
TCP Friendliness and Immunity to Drop-to-Zero	Better than PGMCC and TFMCC	Fig. 4
Multiple Bottleneck Fairness	Comparable with PGMCC and TFMCC	Fig. 6
Slowest Receiver Tracking	Better than PGMCC and TFMCC	Fig. 8
Feedback Suppression	Better than PGMCC and TFMCC	Table II
Performance in Heterogeneous Dynamic Network	Better than PGMCC and TFMCC	Table II

[8] J. C. Lin, S. Paul, "RMTP: A reliable multicast transport protocol", *Infocom 1996*, March 1996

[9] J. Macker, R. Adamson, "A TCP Friendly, Rate-Based Mechanism for Nack-Oriented Reliable Multicast Congestion Control", *Globecom 2001*

[10] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, volume 27, number 3, Jul. 1997.

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

[12] T. Nguyen, K. Nakauchi, M. Kawada, H. Morikawa, T. Aoyama, "Rendezvous Points Based Layered Multicast", *IEICE Trans. Commun.*, Vol. E84-B, No. 12, Dec. 2001.

[13] Jorg Nonnenmacher, Ernst W. Biersack, "Scalable Feedback for Large Groups", *IEEE/ACM Transactions on Networking*, 7(3): 375-386, Jun. 1999.

[14] Jitendra Padhye, Victor Firoiu, Don Towsley, Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *SIGCOMM*, Aug. 1998.

[15] L. Rizzo, "PGMCC: A TCP-friendly Single-Rate Multicast Congestion Control Scheme", *SIGCOMM '00*, Aug '00.

[16] Saswati Sarkar, Tianmin Ren, Leandros Tassiulas, "Achieving Fairness in Multicasting with Almost Stateless Rate Control," Invited Paper Proceedings of SPIE, (Scalability and Traffic Control In IP Networks, II) Vol. 4868, pp. 16-30, ITCOM 2002, Boston

[17] Puneet Sharma, Deborah Estrin, Sally Floyd, Van Jacobson, "Scalable Timers for Soft State Protocols", *Proceedings of IEEE INFOCOM*, Apr. 1997.

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

[19] Brian White et al, "An Integrated Experimental Environment for Distributed Systems and Networks (full report)", *Technical Report of University of Utah, May 2002; Revised version to appear at OSDI 2002, December 2002*

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

[21] http://www.cs.rpi.edu/∼lij6/Research/ormcc/ormcc.html

APPENDIX I
ALGORITHM

A. Source Operations

Variables :

λ : Transmission rate
μ : Throughput rate at congestion (TRAC) in the received CI(μ)
$E(\mu^{cr})$: Average TRAC of the CR
μ_σ^{cr} : Deviation of the CR TRAC
s : Packet size
RTT_{max} : Maximum RTT
RTT_{cr} : RTT between the source and the CR
T^{cr} : CR response time when the bottleneck is fully loaded
$E(T^{cr})$: Average of T^{cr}
T_σ^{cr} : Deviation of T^{cr}
cr_valid : Indicates whether the CR is valid
R : The receiver sending the received CI(μ)

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

Event every RTT_{cr}**:**
if *There is no rate reduction within the recent* RTT_{cr} **then**
$\lambda \leftarrow \lambda + s/RTT^{cr}$
if $\lambda \geq E(\mu^{cr}) + 4\mu_\sigma^{cr}$ **and** *CR checking timer is not running* **then**
Start CR checking timer with time length $E(T^{cr}) + 8T_\sigma^{cr}$
$t \leftarrow$ the current time
endif
endif

Event when the CR checking timer expires:

cr_valid ← **false**

Send packet:
if *cr_valid is* **true then**
Send a packet with real $E(\mu^{cr})$ and μ_σ^{cr}
else
Send a packet with invalid values for $E(\mu^{cr})$ and μ_σ^{cr}
endif

Subroutine : CutRate ()
if λ *has not been cut within the most recent* RTT_{cr} **then**
$\lambda \leftarrow \min(\lambda, 0.75\mu)$
Stop CR checking timer
endif

Event upon receipt of CI(μ):
if R *is CR* **then**
cr_valid ← **true**
if *CR checking time is running* **then**
$\Delta \leftarrow$ current time $- t$
Update $E(T^{cr})$ and T_σ^{cr} with Δ
endif
Update $E(\mu^{cr})$ and μ_σ^{cr} with μ
Update RTT_{cr}
if $RTT_{max} < RTT_{cr}$ **then**
$RTT_{max} \leftarrow RTT_{cr}$
endif
Stop CR checking timer
do CutRate ()
return
endif

/* The CI(μ) is NOT from CR if reach here */
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 *The RTT sample measured by this CI(μ) is larger than* RTT_{cr} **then**
Choose R as the CR
endif
/* NOT in CR grace period */
else if $\mu < E(\mu^{cr}) - \mu_\sigma^{cr}$ **then**
Choose R as the CR
endif

if *CR has been changed at the receipt of this CI(μ)* **then**
cr_valid ← **true**
Update $E(\mu^{cr})$ and μ_σ^{cr} with μ
Update RTT_{cr}
if $RTT_{max} < RTT_{cr}$ **then**
$RTT_{max} = RTT_{cr}$
endif
do CutRate ()
endif

B. Receiver Operations

Variables :
μ : A throughput rate at congestion (TRAC) sample
$E(\mu)$: Average TRAC of this receiver
$E(\mu^{cr})$: Average TRAC of the CR
μ_σ^{cr} : Deviation of the CR TRAC

Event upon receipt of a packet:
if $E(\mu^{cr})$ *and* μ_σ^{cr} *has been changed* **then**
Update the local copy of $E(\mu^{cr})$ and μ_σ^{cr}
endif
if *This packet indicates packet losses* **then**
Meaure μ and update $E(\mu)$
if $E(\mu^{cr})$ *and* μ_σ^{cr} *are invalid* **or** $E(\mu) < E(\mu^{cr}) - \mu_\sigma^{cr}$ **then**
Send CI(μ)
endif
endif

APPENDIX II
THEORETICAL ANALYSIS

A. Capability of Tracking The Slowest Receiver

In this part, we are going to show that an ORMCC flow always track the slowest receiver, i.e. the receiver behind the most congested path. For convenience, we are going to refer the path between the source and the CR as *Representative Path*.

Let's consider a multicast session using ORMCC. Suppose there are N ($N > 1$) different paths on the multicast tree. Let R_i be the receiver behind path i. Without loss of generality, assume R_1 is the current CR. The source will choose another receiver R_j ($j \neq 1$) as the new CR only if R_j sees a lower throughput rate at congestion (TRAC) than that seen by R_1. To see when a R_j will see a lower TRAC on average, first we are going to calculate the TRACs on all paths from 1 to N, given the instantaneous ORMCC sending rate at which a burst of packet losses begins.

For the analysis, we have the following definitions (all i are from 1 to N):

$\lambda_{i,t}$: Instantaneous sending rate of the ORMCC flow at time t on path i.
$\mu_{i,t}$: Instantaneous throughput rate of the ORMCC flow corresponding to $\lambda_{i,t}$.
$\lambda_{i,t}^o$: The sum of instantaneous sending rates of all other flows sharing the bottleneck on path i at time t.
W_i : Bandwidth of the bottleneck on path i.
Q : Buffer size of a bottleneck. Here we assume that all bottlenecks 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.
s : Packet size. We assume that all the packet sizes are equivalent.
RTT_i : RTT of path i.
Δ : The sending rate increment of the ORMCC flow per unit time. $\Delta = s/RTT_1 > 0$.
Δ_i : The sum of the sending rate increments per unit time of all but the ORMCC flows sharing the bottleneck on path i, without packet losses occuring at the bottleneck, assuming all do AIMD. In reality, most likely Δ_i changes randomly, therefore we consider its average value in an aggregate sense. $\Delta_i > 0$.
γ_i : $\gamma_i = \Delta_i/\Delta$.

Moreover, we assume that data are sent bit-by-bit evenly, sending rates are increased continuously, as well as that all packet losses are due to congestion. Also, drop-

tail buffer management is assumed for bottlenecks.[10]

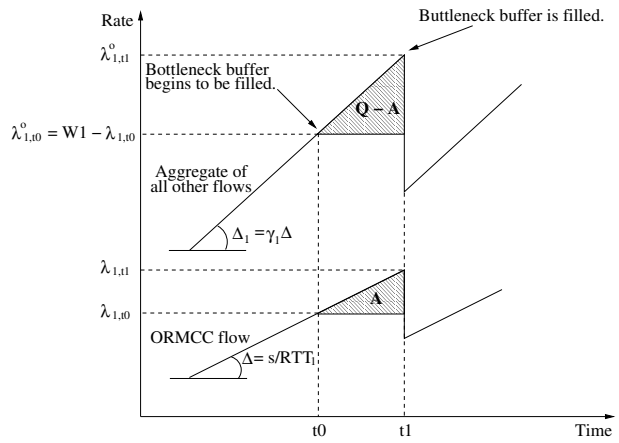


Fig. 12. EVOLUTION OF THE ORMCC FLOW'S SENDING RATE ON THE REPRESENTATIVE PATH

Let's consider path 1 first (Figure 12). Suppose at time t_1, there is a burst of packet losses. The bottleneck queue must be full at this moment. The sum of sending rates of all the flows going through the bottleneck, $\lambda_{1,t_1} + \lambda^o_{1,t_1}$, must be larger than the bottleneck bandwidth W_1. Recall that $\Delta > 0$ and $\Delta_1 > 0$, meaning that without packet losses at the bottleneck, the sum of sending rates keep increasing. Consequently, at an earlier moment t_0, the sending rate sum must be equal to W_1, i.e.,

$$\lambda_{1,t_0} + \lambda^o_{1,t_0} = W_1 \tag{1}$$

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

$$\lambda_{1,t_1} = \lambda_{1,t_0} + (t_1 - t_0)\Delta \Rightarrow \lambda_{1,t_0} = \lambda_{1,t_1} - (t_1 - t_0)\Delta \tag{2}$$

$\lambda^o_{1,t}$ grows by $\Delta_1 = \gamma_1\Delta$ per unit time, therefore,

$$\lambda^o_{1,t_1} = \lambda^o_{1,t_0} + (t_1 - t_0)\Delta_1 = \lambda^o_{1,t_0} + (t_1 - t_0)\gamma_1\Delta \tag{3}$$

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

$$\begin{aligned} \lambda^o_{1,t_1} &= W_1 - (\lambda_{1,t_1} - (t_1 - t_0)\Delta) + (t_1 - t_0)\gamma_1\Delta \\ &= W_1 - \lambda_{1,t_1} + (t_1 - t_0)(1 + \gamma_1)\Delta \end{aligned} \tag{4}$$

We also assume that at t_0, the bottleneck queue size is zero. Since at t_1, the queue is full, the queue is filled by sending rate increments during $[t_0, t_1]$. Recalling that the total sending rate grows by $\Delta + \Delta_1 = (1 + \gamma_1)\Delta$ per unit time, and the assumption of all flows' doing AIMD, we have,

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

Together with (4),

$$\lambda^o_{1,t_1} = W_1 - \lambda_{1,t_1} + \sqrt{2\Delta Q(1 + \gamma_1)} \tag{5}$$

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

Assuming all flows going through the bottlenck have the same priority, since at time t_1 the bottleneck is working at its full load, we know

$$\begin{aligned} \mu_{1,t_1} &= \frac{\lambda_{1,t_1}}{\lambda_{1,t_1} + \lambda^o_{1,t_1}} W_1 \\ &\overset{(5)}{=} \frac{\lambda_{1,t_1}}{1 + \frac{1}{W_1}\sqrt{2\Delta Q(1 + \gamma_1)}} \end{aligned} \tag{6}$$

On any other path j ($j = 2 \ldots N$), since the ORMCC source ignores the congestion indications on this path (Figure 13), the sending rate of the ORMCC 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,

$$\mu_{j,t_1} = \frac{\lambda_{j,t_1}}{1 + \frac{1}{W_j}\sqrt{2\Delta Q(1 + \gamma_j)}} \tag{7}$$

Consider λ_{i,t_1} ($i = 1 \ldots N$). Assume that the sending rate of the ORMCC flow varies between λ_{min} and λ_{max} (Figure 13), then λ_{i,t_1} is a sample value of a random variable Λ_i with sample space as $[\lambda_{min}, \lambda_{max}]$. Assuming Λ_i's are identically distributed, their expected values are the same, i.e. $E(\Lambda_i) = E(\Lambda_j)$ $(i \neq j)$.

Let receiver i be the receiver behind path i. μ_{i,t_1} $(i = 1 \ldots N)$ is the TRAC measured at receiver i. According to (7), μ_{i,t_1} is a function of λ_{i,t_1}, and thus is a random sample. Denote the corresponding random variable as U_i. Assuming that W_i, Q, s are constant, and that γ_i and RTT_1 in steady state have small deviations and thus can be treated as constant, we have,

$$U_i = \frac{\Lambda_i}{1 + \frac{1}{W_i}\sqrt{2\Delta Q(1 + \gamma_i)}}$$

$$E(U_i) = \frac{E(\Lambda_i)}{1 + \frac{1}{W_i}\sqrt{2\Delta Q(1 + \gamma_i)}}$$

As designed in ORMCC, for $j = 2 \ldots N$, only upon detection of $E(U_j) < E(U_1)$ (the average TRAC of the current CR receiver 1) will receiver j send congestion indication (CI) packets back to the source, which then update the congestion representative (CR) to receiver j. From the expression of $E(U_i)$ above, we have,

$$\begin{aligned} & E(U_j) < E(U_1) \\ \Leftrightarrow \quad & \frac{E(\Lambda_j)}{1 + \frac{1}{W_j}\sqrt{2\Delta Q(1 + \gamma_j)}} < \frac{E(\Lambda_1)}{1 + \frac{1}{W_1}\sqrt{2\Delta Q(1 + \gamma_1)}} \\ \Leftrightarrow \quad & \frac{W_j}{\sqrt{1 + \gamma_j}} < \frac{W_1}{\sqrt{1 + \gamma_1}} \qquad \text{since } E(\Lambda_j) = E(\Lambda_1) \end{aligned} \tag{8}$$

We can see that $W_i/\sqrt{1 + \gamma_i}$ $(i = 1 \ldots N)$ 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, $W_i/\sqrt{1 + \gamma_i}$ has a lower value; if more flows are sharing a bottleneck, the sum of their per-unit-time

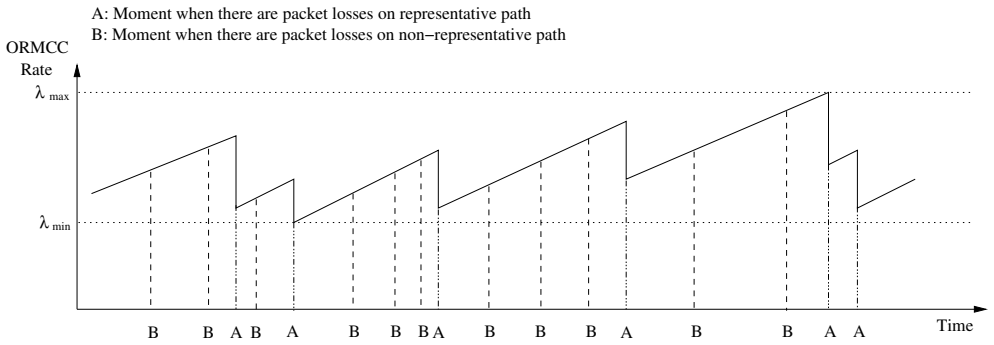


Fig. 13. THE ORMCC SOURCE ONLY CONSIDERS THE CONGESTIONS ON THE REPRESENTATIVE PATH FOR RATE ADAPTATION

rate increments Δ_i is higher, $\gamma_i = \Delta_i/\Delta$ is then larger, which in turn also makes $W_i/\sqrt{1+\gamma_i}$ lower. Therefore, (8) 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 $E(U_j)$, and will send CI(μ)s back to the source, making the source change CR. Namely, *an ORMCC flow always tracks the slowest receiver.*

B. TCP-Friendliness on Representative Path

By representative path, we mean the path which the congestion representative (CR) is behind. In the following, we are going to show that an ORMCC flow is friendly to a TCP flow on the representative path, by showing that they get approximately equal share of the bottleneck bandwidth, with the assumption that their RTT estimations and packet sizes are the same. More strictly speaking, we want to show that, with proper choice of rate reduction factor β for ORMCC, V_t^{MCC}/V_t^{TCP} oscillates around 1, where V_t^{MCC} and V_t^{TCP} denote the sending rates of the TCP flow and ORMCC flow at time t respectively. Those two flows are assumed to be the only flows on the representative path. A sample of the rate evolution is given in Figure 14.

Like other TCP throughput analysis papers [14] [10] 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 $1/N$ packet upon the receipt of per ACK, where N is the current congestion window size. A TCP source transmit all the packets in its congestion window in one RTT, therefore, the window grows by 1 packet per RTT, [11] which corresponds to the fact that its sending rate is increased by s/RTT per RTT, where s is the packet size. An ORMCC source increases its sending rate at the same pace, as covered in scheme description. At packet

[11] We assume that a TCP receiver sends an ACK per received packet.

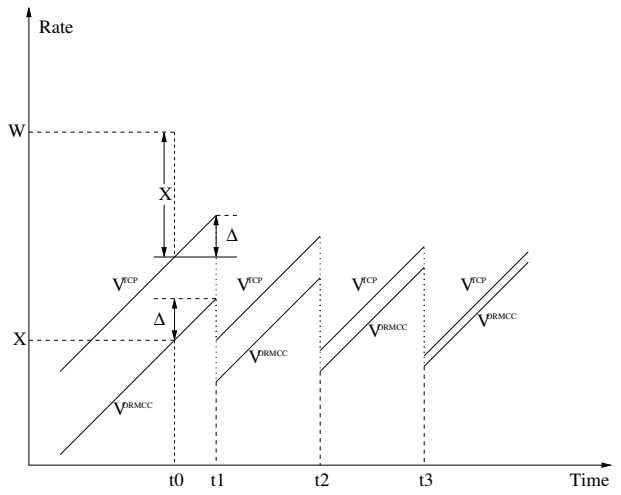


Fig. 14. EVOLUTION OF THE SENDING RATES OF TCP AND ORMCC FLOWS

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. Let W be the bottleneck bandwidth. It is obvious that packet losses can occur only if $V_t^{TCP} + V_t^{MCC} \geq W$. Suppose some packets are lost and both flows reduce their transmission rates at t_1 (Figure 14). Before the losses, since both V_t^{TCP} and V_t^{MCC} keep increasing, there must be a moment t_0 when $V_{t_0}^{TCP} + V_{t_0}^{MCC} = W$. For short, let $V_{t_0}^{MCC} = X$, then $V_{t_0}^{TCP} = W - X$. For the first step of analysis, we will show that with appropriate β,

$$\begin{cases} X < W - X \Rightarrow X/(W-X) < V_{t_1}^{MCC}/V_{t_1}^{TCP} \\ X > W - X \Rightarrow X/(W-X) > V_{t_1}^{MCC}/V_{t_1}^{TCP} \end{cases} \tag{9}$$

Let the moment just before the rate reduction at t_1 be t_1'. Because the TCP and ORMCC flows share the same path, we 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.

$$V_{t_1'}^{TCP} = W - X + d$$

After a reduction by half,

$$V_{t_1}^{TCP} = \frac{V_{t_1'}^{TCP}}{2} = \frac{W - X + d}{2}$$

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

$$V_{t_1'}^{MCC} = X + d$$

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

$$\frac{V_{t_1'}^{MCC}}{V_{t_1'}^{MCC} + V_{t_1'}^{TCP}} W = \frac{X + d}{W + 2d} W$$

According to the rate adaptation policy of ORMCC,

$$V_{t_1}^{MCC} = \beta \frac{X + d}{W + 2d} W$$

Therefore,

$$\frac{V_{t_1}^{MCC}}{V_{t_1}^{TCP}} = \beta \frac{X + d}{W + 2d} W \Big/ \frac{W - X + d}{2}$$

Now let's compare $X/(W - X)$ and $V_{t_1}^{MCC}/V_{t_1}^{TCP}$.

$$\begin{aligned} & \frac{X}{W - X} - \frac{V_{t_1}^{MCC}}{V_{t_1}^{TCP}} \\ = \; & \frac{2}{W - X + d} \\ & \cdot \left[\left(\frac{1}{2} - \frac{\beta W}{W + 2d} \right) X + \left(\frac{X}{2(W - X)} - \frac{\beta W}{W + 2d} \right) d \right] \end{aligned} \tag{10}$$

Since $W > X$ and $d \geq 0$, $2/(W - X + d) > 0$, and the positivity of (10) is decided by its second factor between square brackets. If we choose a value for β carefully so that,

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

The second factor of (10) becomes

$$0 \cdot X + \frac{d}{2} \left(\frac{X}{W - X} - 1 \right) = \frac{d}{2} \left(\frac{X}{W - X} - 1 \right) \tag{11}$$

It is easily seen that, if $X > W - X$, (11) > 0 so that (10) > 0; while if $X < W - X$, (11) < 0 so that (10) < 0. That is exactly what we want for (9).

With (9) eastablished, we can go further. Assume that at t_i, $i = 1 \ldots \infty$, both the TCP flow and the ORMCC flow reduce their rates (Figure 14), and their rates after reduction are $V_{t_i}^{TCP}$ and $V_{t_i}^{MCC}$ respectively. Also assume that $t_{i,0}$ is the closest moment before t_i so that $V_{t_{i,0}}^{TCP} + V_{t_{i,0}}^{MCC} = W$. Recall the meanings of X and $W - X$, (9) actually indicates that,

$$\begin{cases} V_{t_{i,0}}^{MCC} < V_{t_{i,0}}^{TCP} \Rightarrow V_{t_{i,0}}^{MCC}/V_{t_{i,0}}^{TCP} < V_{t_i}^{MCC}/V_{t_i}^{TCP} \\ V_{t_{i,0}}^{MCC} > V_{t_{i,0}}^{TCP} \Rightarrow V_{t_{i,0}}^{MCC}/V_{t_{i,0}}^{TCP} > V_{t_i}^{MCC}/V_{t_i}^{TCP} \end{cases} \tag{12}$$

Let's consider the situation that $V_{t_{i,0}}^{MCC} < V_{t_{i,0}}^{TCP}$ first, which is shown in Figure 14. Note that for any i,

$$V_{t_{i+1,0}}^{MCC} = V_{t_i}^{MCC} + A, \;\; V_{t_{i+1,0}}^{TCP} = V_{t_i}^{TCP} + A \quad A > 0$$

So,

$$\frac{V_{t_i}^{MCC}}{V_{t_i}^{TCP}} < \frac{V_{t_i}^{MCC} + A}{V_{t_i}^{TCP} + A} = \frac{V_{t_{i+1,0}}^{MCC}}{V_{t_{i+1,0}}^{TCP}} \overset{(12)}{<} \frac{V_{t_{i+1}}^{MCC}}{V_{t_{i+1}}^{TCP}}$$

Similarly, if $V_{t_{i,0}}^{MCC} > V_{t_{i,0}}^{TCP}$,

$$\frac{V_{t_i}^{MCC}}{V_{t_i}^{TCP}} > \frac{V_{t_{i+1}}^{MCC}}{V_{t_{i+1}}^{TCP}}$$

As the result, if the ORMCC flow rate is less than that of the TCP flow, it will grow until it exceeds the latter; likewise, if the ORMCC flow rate is more, it will get less and less until it is below the TCP flow rate. Hence, V_t^{MCC}/V_t^{TCP} oscillates around 1. In conclusion, we say that ORMCC is approximately TCP friendly, given that the rate reduction factor β is properly chosen.

With regard to the value of rate reduction factor β, recall that above in the analysis, we need to have that

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

Since $d \geq 0$, β needs to have a value greater than 0.5. Consider the fact that TCP uses ACKs to measure RTTs. It can have lower RTT estimation than that of ORMCC which uses NAKs for this purpose, as we discussed in Section III-A.6. Thus, TCP can increase sending rate faster than ORMCC. For compensation, ORMCC 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.

C. 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. However, in nature, the source in ORMCC 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 ORMCC. In more details, if a receiver other than the current congestion representative (CR) sees a packet loss rate lower or equal to that by CR, it won't send CI(μ)s back to the source. The source won't see any CI(μ)s from it, thus of course won't reduce the transmission rate. Even if the source gets CI(μ)s 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 CI(μ)s from all other receivers. Consequently, we can say that ORMCC is immune to drop-to-zero.

D. Effectiveness of Feedback Suppression

Without support from internal nodes, which is the situation that we assume for reality, most multicast feedback suppression schemes (e.g. [6], [17], [20], [7], [13]) use random timers for delaying receivers' feedback before sending them. However small it is, there is some feedback latency 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 [7]. Moreover, we are going to show below that in ORMCC, the total number of feedbacks (i.e. CI(μ)s) sent to the source by all receivers in a multicast session, is independent of the total number of receivers. Instead, it depends on the switching frequency of the most congested bottleneck, as well as the number of receivers behind the new most congested bottleneck, plus the minimum RTT between them 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[12]. To begin the calculation, the following notations are needed[13].

N : Total number of receivers behind the new most congested bottleneck (MCB).

R_i : Receiver i behind the new MCB. ($i = 1 \dots N$)

RTT_i : RTT between the source and R_i.

RTT_i^f : Forwarding (downstream) part of RTT_i.

RTT_{min} : The minimum of all RTT_i's.

p : Packet loss rate seen by receivers behind the new MCB.

v : Average transmission rate of the ORMCC flow.

Whenever there is a new MCB, according to the previous discussion of ORMCC's following the most congested path, only those receivers behind the new MCB will send CI(μ)s back to the source, and all of them except one will stop sending CI(μ)s once one of them is chosen as the new CR. More specifically, the source will first see the CI(μ)s from the receiver with RTT_{min}, then change CR and tell all receivers of the change. For any R_i except the new CR, the duration of sending CI(μ)s 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 the new CR. According to Figure 15, $t_{i_1} - t_{i_0} = RTT_i^f + RTT_{min} - RTT_i^f = RTT_{min}$. Therefore, before a new CR is decided, the number of CI(μ)s sent from this receiver i is thus $pvRTT_{min}$, and the total number of CI(μ)s sent by all receivers behind the new MCB is,

$$\sum_{i=1}^{N} pvRTT_{min} = vpN \cdot RTT_{min}$$

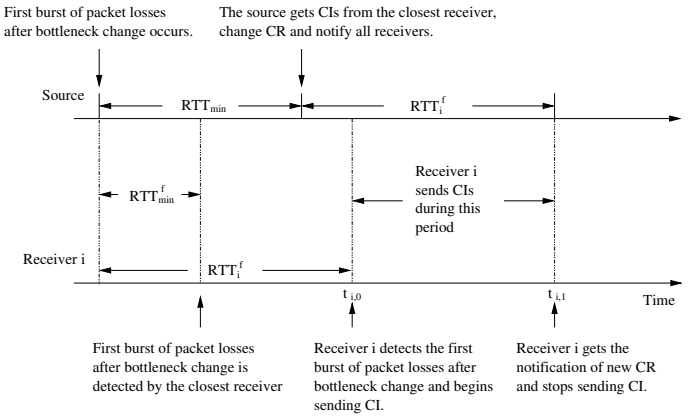


Fig. 15. FEEDBACK SUPPRESSION

Once a new CR is decided, only one receiver, namely the new CR, will send CI(μ)s. Let's call the period between two successive MCB switchings *MCBSP (MCB switching period)*. During a MCBSP of length t, the total number of CI(μ)s sent to the source is, assuming $t \geq RTT_{min}$,

$$vpN \cdot RTT_{min} + vp(t - RTT_{min}) = vp(t + (N-1)RTT_{min})$$

If a Poisson process with parameter λ[14] is assumed for MCB switching, for a multicast session of duration T, on average, the total number of CI(μ)s transmitted is approximately,

$$\lambda T \cdot vp\left(\frac{1}{\lambda} + (N-1)RTT_{min}\right) \tag{13}$$

We can see that, for a ceratin T,

$$\begin{aligned} & \lim_{\lambda \to 0} \lambda T \cdot vp\left(\frac{1}{\lambda} + (N-1)RTT_{min}\right) \\ = \quad & vpT + \lim_{\lambda \to 0} \lambda vpT(N-1)RTT_{min} = vpT \end{aligned} \tag{14}$$

That means, if during a multicast session, there is no MCB switching, the total number of CI(μ)s transmitted is approximately equal to the number of CI(μ)s sent from a single receiver behind the MCB. To make it clearer: if the MCB does not change during a multicast session, the volume of feedback is on the same level of unicast feedback!

Also, from (13), we find that the total number of transmitted CI(μ)s is independent of the total number of receivers in a multicast session (Note that N is (13) is not the total number of receivers but the number of receivers behind the MCB). It depends on how fast MCB switches

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

[13] 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.

[14] Considering the reality, we can assume that MCB switching does not occur too frequently and $1/\lambda \geq RTT_{min}$.

and the amount of receivers behind the new MCB, as well as the smallest RTT between those receivers and the source. Usually, MCB switches only once in many multiple RTT_{min}'s, and the amount of receivers behind the new MCB is much less than the overall number. Moreover, RTT_{min} is almost a negligible duration. Consequently, our feedback suppression mechanism is effective.

Finally, we must say that due to measurement errors in reality, the total number of CI(μ)s sent can be a little higher than what we have derived here. However, the difference won't be significant.