ORMCC - Multicast Congestion Control Based on Output Rate Feedback

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

***Abstract*— There are two categories of multicast congestion control: multi-rate and single-rate. The former can adapt the transmission rate to heterogeneous receivers but requires layered transmission. The latter is simple to implement and deploy, and thus at least suitable for some situations. In this article, a new single-rate scheme ORMCC is proposed. It requires very simple support from receivers, which distinguishes it from other schemes in this category, e.g. PGMCC [11], TFMCC [12] and MDP-CC [8]. Simulations on *ns-2* [1] show that the scheme is TCP-friendly and does not suffer from drop-to-zero problem.**

***Index Terms*— Multicast, congestion control, single-rate, drop-to-zero, TCP friendliness**

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; 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 as much data as it can.

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 [9]. Aggregated multicast trees [4] 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 to routers. On the contrary, the single-rate category is easy to implement and deploy, because it does not require support from intermediate nodes. The recently published work includes PGMCC [11], TFMCC [12] and MDP-CC [8]. Although such schemes do not scale as well as multi-rate ones, they are at least suitable for some situations, e.g. the multicast in a not-so-heterogeneous environment, or bulk data transfer without concerns over delay. With network support [7], we can also emulate multi-rate schemes by deploying single-rate schemes on intermediate nodes.

The scheme we propose in this paper falls into the single-rate category. The key observation in this paper is that, with some concise and easily measured yet more meaningful feedback information from receivers, rather than with one-bit feedbacks such as simple ACKs or NAKs, we can reduce both the state and computation complexity at source to $O(1)$.

The feedback information is the output rate at receivers' end (μ in Figure 1), sent in *Congestion Indication (CI)* packets[1]. Using this information, the source selects the slowest receiver as the *Congestion Representative (CR)* and adapts the transmission rate correspondingly. Notice that in our scheme, (1) CI carrying receiver-end output rate is *the only support required from receivers*,[2] and (2) The traffic between source and receivers is conventional, i.e. data packets from source and CIs from receivers (See Figure 1). This simplicity distinguishes our work from previous ones like PGMCC [11], TFMCC [12] and MDP-CC [8]. Recall that all these three schemes choose representatives according to a metric based on TCP throughput formula [10]. Besides, they all demand additional traffic: (1) PGMCC maintains ACK traffic between source and ACKER, (2) TFMCC requires additional traffic for RTT measurement at receiver side, and (3) MDP-CC needs control messages multicasted to receivers.[3] Because our scheme is based on output rates, we named it ORMCC. Note that it handles congestion control only, and does not take charge of data reliability.

As usual, we need to deal with the *Drop-to-Zero* and *TCP-friendliness* problem. The former can happen if the transmission rate is reduced more than necessary and

[1] CI packets are those packets sent at least once per RTT by receivers to source indicating packet losses and thus congestion. They can be, but not necessarily limited to, NAKs.

[2] As in other schemes, sequence number of lost packets are also put into CIs for RTT estimation. Since it is conventional, we don't list it as a special support from receivers.

[3] MDP-CC also maintains a pool of representative candidates while our scheme does not.

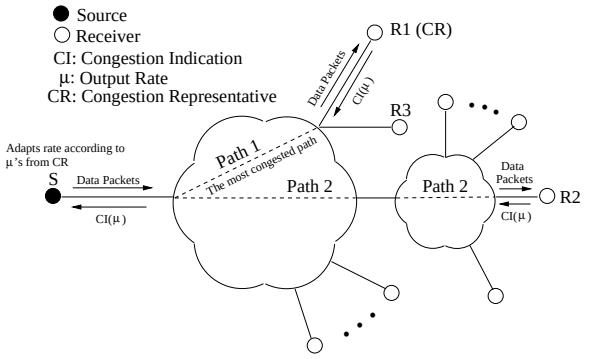


Fig. 1. ORMCC MODEL

beaten down, when multiple paths in the multicast tree experience asynchronous congestion. The latter is whether the multicast flow can compete for bandwidth fairly with TCP flows on a common bottleneck. We have also done some initial study on the performance with lower rate of feedback (at most one CI per RTT) when feedback suppression is needed to avoid implosion. In the following sections, we will first describe the scheme, followed by simulation results. At the end, we will briefly discuss our future research.

II. ORMCC

In single-rate multicast congestion control, the transmission rate should be set to the fair rate on the most congested path (e.g. $Path1$ in Figure 1). Such a path can be allocated by correctly selecting the slowest receiver (e.g. $R1$ in Figure 1), which in our scheme is detected by using output rate feedbacks from receivers. The whole scheme will be described in details in this section. In the following context, we are going to refer to the selected slowest receiver as *Congestion Representative (CR)*.

A. Issues to Solve and Solutions

Even if the motivation of selecting CR is straightforward, there are two issues to solve: (1) How do we keep track of CR? The CR should be valid, i.e. be the slowest receiver, and the choice of CR should not be switched back and forth too often, since that may affect the rate adaption negatively. (2) How do we adapt the transmission rate? There should not be Drop-to-Zero problem, and the scheme should be TCP friendly. To solve these two issues, we have to provide solutions for the following sub-problems.

What information is required from receivers?

From our previous work on multicast congestion avoidance [5], we realize that output rate at receiver side can convey enough information and is easy to measure. That is the only support from receivers. The way for a receiver to measure the output rate in our implementation is to calculate $n \cdot s/t$, where n is the number of packets, s is the packet size, t is the time needed to get n packets. $n = 4$ in our simulations.

How to refine the choice of CR?

The network condition can change, and thus the current CR may no longer be the slowest receiver. Or, the initial CR chosen upon the first CI's arrival (Algorithm line 8 $\sim$ 12) may not be good. If that happens, we will have to switch to another suitable receiver. For example, in Figure 1, if $Path2$ becomes the most congested path, $R2$ should be chosen as the new CR. (For this problem, we assume that no current or potential CR will go offline.)

We check two situations: (1) *The most congested path improves and is no longer the most congested.* (2) *Another path worsens and becomes the new most congested path.* For that, μ_{cr}, μ_{avg} and σ_{avg} are maintained at source. μ_{cr} is the EWMA (exponentially weighted moving average) of the μ's (output rates) from the current CR, μ_{avg} and σ_{avg} are respectively the EWMA and estimated standard deviation of the μ's from all receivers. If $\mu_{cr} > \mu_{avg} + 2\sigma_{avg}$, either the first or the second situation is true, and we are ready to switch CR upon receipt of μ from the new slowest receiver (Algorithm line 13 $\sim$ 17, 22 $\sim$ 27). If $\mu < \mu_{avg} - 4\sigma_{avg}$, the second situation is true, CR is switched right away (Algorithm line 28 $\sim$ 35). The $4\sigma_{avg}$ deviation from mean implies that the new feedback should be significantly lower in order to result in CR switching.

How to keep the CR choice stable if there are multiple candidates for CR?

There may be multiple slowest receivers under certain situations. All of them are qualified for CR (e.g. $R1$ and $R3$ in Figure 1). In spite of that, after we pick one of them as CR, as long as we follow the rules in the above problem, the choice of CR should be stable.

How to update CR if the current CR goes offline?

If the current CR goes offline, the source will no longer receive CIs from it and be unaware of congestion. Therefore, the source should be able to detect that situation and switch CR. We set a response interval threshold I in terms of packets. According to the throughput formula in [10], $\tilde{\Lambda} = s/(RTT(\sqrt{2p/3} + 12\sqrt{3p/8}p(1+32p^2)))$ (where $\tilde{\Lambda}$ is the throughput rate, s is the packet size, p is the loss event rate), we calculate [4] I as $I = 1/p = (\tilde{\Lambda} \cdot RTT/s)^2$, omitting some negligible terms. If there has not been CIs from the current CR within the most recent interval of length I, it is deemed offline and the source will switch CR (Algorithm line 36 $\sim$ 42). The detailed derivation can be found in our technical report [6].

How to measure RTT?

[4] In our algorithm, we use $\Lambda + 3\sigma_\Lambda$ in place of $\tilde{\Lambda}$. Please refer to the algorithm for the definitions of Λ and σ_Λ.

When a receiver gets a packet of sequence number k, if it detects packet losses, it sends a CI packet to the source with k. Upon the arrival of this CI packet, the source measures a RTT sample RTT_s as the difference between the current time and the transmission time of the packet with sequence number k. With RTT_s, the RTT is updated as $RTT = 7/8 \cdot RTT + 1/8 \cdot RTT_s$.

How to reduce the transmission rate?

Upon receipt of μ's from the CR, the transmission rate λ is updated as $\lambda \leftarrow \min(\lambda, \beta\mu)$, and there is at most one rate reduction per RTT. β can affect the fairness between flows. In our simulations, $\beta = 0.75$ works well.

How to increase the transmission rate?

Usually, we increase the rate by $s/SRTT$ per SRTT where s is the data packet size and $SRTT = RTT + 4\sigma$. Since the RTT samples measured by CIs include the full queue delay, the RTT is thus actually the upper-bound of the real round trip time. Therefore, we use RTT instead of SRTT for the rate increment purpose.

B. Algorithm

With the clues above, we designed the algorithm at the source. It is executed whenever the source gets a CI.

Variables:

- bad_cr : Binary variable indicating whether the current CR is no longer qualified.
- μ : Output rate in CI
- R : The receiver which sent this CI
- μ_{avg} : EWMA of all μ's
- σ_{avg} : Standard deviation of all μ's
- μ_{cr} : EWMA of μ's from CR
- λ : Transmission rate
- λ^o : The transmission rate at last reduction
- Λ : EWMA of the throughput
- σ_Λ : Standard deviation of the throughput
- I : Expected largest interval between two neighboring CIs from CR

Subroutine: CutRate ()

if *The transmission rate has been cut within the most recent RTT* **or** $\lambda \leq \beta\mu$ **then**
  Return
else
  $err \leftarrow (\lambda + \lambda^o)/2 - \Lambda$
  $\Lambda \leftarrow \Lambda + \gamma \cdot err$ ($\gamma = 0.5$ in our implementation)
  $\sigma_\Lambda \leftarrow \sigma_\Lambda + \gamma(|err| - \sigma_\Lambda$
  $I \leftarrow ((\Lambda + 3\sigma_\Lambda) \cdot RTT/s)^2$
  $\lambda \leftarrow \beta\mu$ ($\beta = 0.75$ in our implementation)
  $\lambda^o \leftarrow \lambda$
endif

Main Algorithm:

$err \leftarrow \mu - \mu_{avg}$
$\mu_{avg} \leftarrow \mu_{avg} + \alpha \cdot err$ ($\alpha = 0.125$ in our implementation)
$\sigma_{avg} \leftarrow \sigma_{avg} + \alpha(|err| - \sigma_{avg})$
if *The CI is from the current CR* **then**
  $err \leftarrow \mu - \mu_{cr}$
  $\mu_{cr} \leftarrow \mu_{cr} + \alpha \cdot err$
endif
if *The CI is the very first one* **then**
  Set CR to R
  Do CutRate ()
  Return
endif
if $\mu_{cr} \leq \mu_{avg} + 2\sigma_{avg}$ **then**
  Reset bad_cr flag
else
  Set bad_cr flag
endif
if *The CI is from the CR* **then**
  Do CutRate ()
  Return
endif
/* Now the CI is NOT from the CR */
if *bad_cr flag is set* **and** $\mu \leq \mu_{avg}$ **then**
  Update CR to R
  $\mu_{cr} \leftarrow \mu$
  Do CutRate ()
  Return
endif
if $\mu \leq \mu_{avg} - 4\sigma_{avg}$ **then**
  if $\mu < \mu_{cr}$ **or** *with probability q* **then**
    Update CR to R
    $\mu_{cr} \leftarrow \mu$
    Do CutRate ()
    Return
  endif
endif
if *There has no CI from the current CR within the most recent interval* I **then**
  Update CR to R
  $\mu_{cr} \leftarrow \mu$
  Do CutRate ()
  Return
endif

III. Simulation Results

We investigated the behavior of our scheme under controlled situations by running simulations on *ns-2* [1]. The simulations include (1) Simple Multicast Configuration, (2) Simple Multicast Configuration with Changing Bottlenecks (with normal and lower CI rates), (3) Multiple Bottlenecks (Linear Network), (4) Drop-to-Zero Avoidance Test, (5) TCP-Friendliness Test (with drop-tail and RED queues). In those simulations, the data packet size is 1000 bytes, the bottleneck buffer size is 50K bytes, the initial RTT is 200 milliseconds.

A. Simple Multicast Configuration

This simulation on topology (Figure 2 (a)) tests the basic behavior of our scheme. At time 0, there is only one multicast flow, with the source on Node 1, two receivers on Node 2 and 3 respectively. At 50th and 100th second respectively, two multicast flows are added with the same source-receiver set. In the result (Figure 3 (a)), the average rate between time [0, 50) is around 0.9Mbps, those between [50, 100) are around 0.9 / 2 = 0.45 Mbps, and those

between [100, 400) are around 0.9 / 3 = 0.3 Mbps. Conclusively, the multicast flows shared the bottleneck fairly.

B. Simple Multicast Configuration with Changing Bottlenecks

We ran this simulation to verify that our scheme can detect CR's absence and switch CR in time. Again we used the topology in (Figure 2 (a)), but varied the bandwidths according to the following table. (Link 1 is between Router and Node 2, Link 2 is between Router and Node 3). It is equivalent to the scenario that: at time 0, only Node 2 is in the multicast group; at 50th second, Node 3 joins and Node 2 leaves; at 100th second, Node 2 joins and Node 3 leaves; at 150th second, Node 3 joins.

	[0,50)	[50,100)	[100,150)	[150,400]
Link 1	0.9Mbps	3Mbps	0.9Mbps	0.9Mbps
Link 2	3Mbps	0.9Mbps	3Mbps	0.9Mbps

The vertical lines in Figure 3 (b) show that the source switched CR at around 60th and 110th second as expected. There was no CR switch after 150th second, which is desired because the current CR is enough to convey the congestion information for all the bottlenecks and it is not necessary to switch. Note that because it took time to detect CR's absence, the source reacted to congestion a little late. However, as shown in the figure, the over-congestion is insignificant and transient.

We also did the simulation on this configuration for a slightly modified scheme. In that version, each receiver sent at most one CI per RTT, resulting in over 90% less CIs. The modification does not affect the performance much, as shown in Figure 3 (c).

C. Multiple Bottlenecks (Linear Network)

Different notions of fairness define how the bottleneck bandwidth is shared by through traffic. To check the fairness our scheme can achieve, we ran a simulation on Figure 2 (b). RED queues are used on the routers to eliminate the effect of RTT estimation. There is one multi-receiver multicast flow from Node 1 to Node 4 and 5, two uni-receiver multicast flows from Node 2 to Router 2 and from Node 3 to Node 4 respectively. *Proportional fairness* implies that the long (multi-receiver multicast) flow should get one-third of the bottleneck bandwidth whereas *Max-min fairness* suggests a share of one-half. The average rates [5] in Figure 4 show that in our scheme, the long flow got a share between that suggested by *Proportional fairness* 0.33 Mbps and that by *Max-min fairness* 0.5 Mbps.

[5] Average rate at time t = $1/t \cdot$ The data sent between time 0 and t.

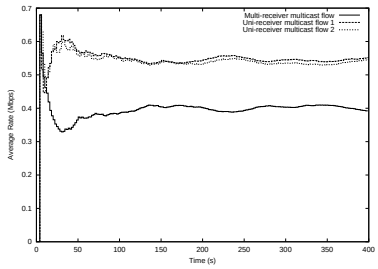


Fig. 4. MULTIPLE BOTTLENECKS SIMULATION RESULT

D. Drop-to-Zero Avoidance Test

Our scheme won't reduce the transmission rate more than necessary at the presence of asynchronous congestion and is thus immune from *Drop-to-Zero* problem. The simulation was run on the star topology Figure 2 (c). There is a background traffic flow of uni-receiver multicast between each pair of Node i and Node $N + i$ ($i = 1 \ldots N, N = 16$). Half of the links between Router and Node $N + 1$ to $2N$ have bandwidth of 0.9Mbps, the other half 1Mbps. There is also a multi-receiver multicast flow with its source on Node 1 and receivers on Node $N + 1$ to $2N$. It is expected that the multi-receiver flow shares the 0.9Mbps bottlenecks *equally* with those uni-receiver flows. The result (Figure 5) confirms that.

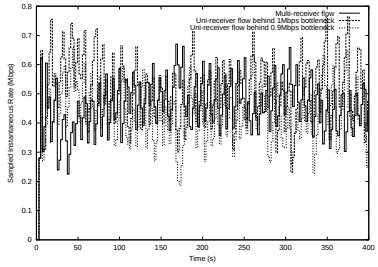


Fig. 5. DROP-TO-ZERO AVOIDANCE TEST RESULT (Uni-receiver flows are randomly chosen. Sample interval = 2 seconds.)

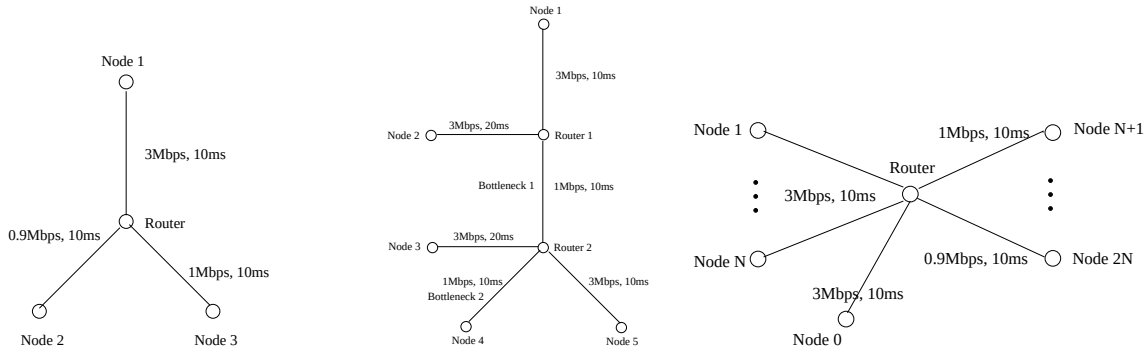
E. TCP-Friendliness Test

As mentioned in the introduction, our scheme should compete for bandwidth fairly with TCP. Again we used the star topology (Figure 2 (c)), but replaced uni-receiver multicast flows with TCP flows. We ran two simulations, one with drop-tail queues on routers, the other with RED [6]. Both results in Figure 6 and Figure 7 show that the multicast flow and TCP flows preformed very closely. The RED result is a little better because the random early drop lets our scheme get CIs before the router queue is full and thus have smaller RTT estimation.

IV. CONCLUSIONS

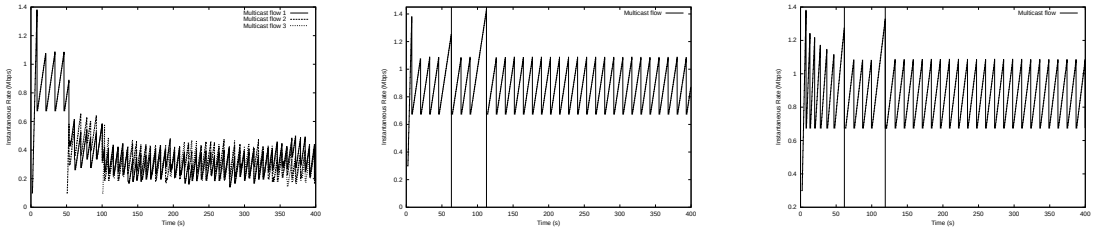
We have presented a single rate source-based multicast congestion control scheme. The only support from receivers is output rates at their ends. Based on the output

[6] For RED, the minimal and maximum threshold are 5 and 15 respectively, and the queue weight is 0.002.



(a) Simple Multicast Configuration (b) Multiple Bottlenecks (c) Star Topology

Fig. 2. SIMULATION TOPOLOGIES



(a) Basic (No Changing Bottlenecks) (b) With Changing Bottlenecks (Normal CI rate) (c) With Changing Bottlenecks (Lower CI rate)

Fig. 3. SIMPLE MULTICAST SIMULATION RESULTS

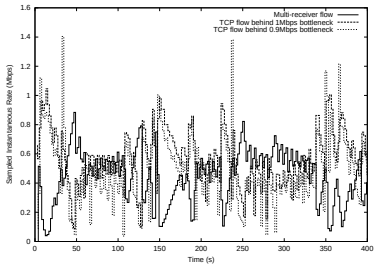


Fig. 6. TCP-FRIENDLINESS TEST (DROP-TAIL) RESULT (TCP flows are randomly chosen. Sample Interval = 2 seconds.)

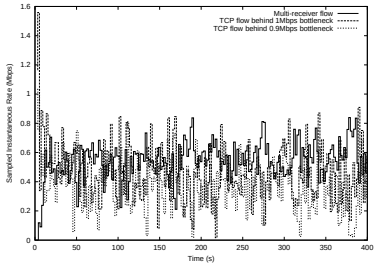


Fig. 7. TCP-FRIENDLINESS TEST (RED) RESULT (TCP flows are randomly chosen. Sample interval = 2 seconds.)

rate feedbacks, the source chooses one of the slowest receiver as *Congestion Representative (CR)*, then adapts the transmission rate correspondingly. We have designed several mechanisms to keep the CR choice stable and valid, and checked the performance of the whole scheme with simulations. The simplicity and the way to choose CR distinguish our scheme from previous work e.g. PGMCC [11], TFMCC [12] and MDP-CC [8]. Our scheme handles congestion control only, thus can be used in combination with other (e.g. reliability) schemes. Due to its simplicity, it can also be potentially deployed on gateways to separated a multicast tree into different segments, each of which can have its own rate, so that heterogeneous receivers may get data at their individually desired rates.

REFERENCES

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

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

[5] J. Li, S. Kalyanaraman, "MCA: A Rate-based End-to-end Multicast Congestion Avoidance Scheme", *ICC 2002*, April 2002.

[6] J. Li, S. Kalyanaraman, "ORMCC - Multicast Congestion Control Based on Explicite Rate Feedback", *Technical Report of CS, RPI*, 2002. Available at http://www.cs.rpi.edu/∼lij6/Research /my_papers/ormcc-tr.ps.gz.

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

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

[9] 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.

[10] J. Padhye, V. Firoiu, D.F. Towsley and J.F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation", *IEEE/ACM Transactions on Networking*, 8(2): 133-145, April 2000.

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

[12] J. Widmer, M. Handley, "Extending Equation-based Congestion Control to Multicast Applications", *SIGCOMM 2001*, Aug 2001