SBQ: A Simple Scheduler for Fair Bandwidth Sharing Between Unicast and Multicast Flows

Fethi Filali and Walid Dabbous

INRIA, 2004 Route des Lucioles, BP-93
06902 Sophia-Antipolis Cedex, France
{filali, dabbous}@sophia.inria.fr

Abstract. In this paper, we propose a simple scheduler called SBQ (Service-Based Queuing) to share the bandwidth fairly between unicast and multicast flows according to a new definition of fairness referred as *the inter-service fairness*. We utilize our recently proposed active queue management mechanism MFQ (Multicast Fair Queuing) to fairly share the bandwidth among all competing flows in the multicast queue.
The simulation results obtained for very heterogeneous sources and links characteristics suggest that, on the one hand, our scheduler achieves the expected aggregated bandwidth sharing among unicast and multicast service, and on the other hand the multicast flows remain TCP-friendly.

1 Introduction

It is widely accepted that one of the several factors inhibiting the usage of the IP multicast is the lack of good, deployable, well-tested multicast congestion control mechanisms.

The precise requirements for multicast congestion control are perhaps open to discussion given the efficiency savings of multicast, but it is well known that a multicast flow is acceptable if it achieves no greater medium-term throughput to any receiver in the multicast group than would be achieved by a TCP flow between the multicast sender and that receiver. Such requirement can be satisfied either by a single multicast group if the sender transmits at a rate dictated by the lowest receiver in the group, or by a layered multicast scheme that allows different receivers to receive different numbers of layers at different rate.

The IETF orchestrated a very strong guideline for developing a TCP-friendly multicast congestion control scheme [14] regardless of the number of receivers in the multicast session. It is well-known that multiplicative decrease/linear increase congestion control mechanisms, and in particular TCP, lead to proportional fairness [9]. However, the authors of [2] have proven that if we treat the multicast flow as if it were a unicast flow, then the application of Kelley's model [9] shows that the larger the multicast group the smaller its share of the proportional bandwidth it would get.

In this paper, we develop a novel approach that helps multicast congestion control mechanisms to fairly exist with TCP protocol. Our approach is based

B. Stiller et al. (Eds.): QofIS/ICQT 2002, LNCS 2511, pp. 78–89, 2002.

on a new fairness notion, *the inter-service fairness*, which is used to share the bandwidth fairly between unicast and multicast services. In our fairness definition, the aggregated multicast traffic should remain *globally* TCP friendly in each communication link. In other words, the aggregated multicast average rate should not exceed the sum of their TCP-friendly rates. This approach allows ISPs to define their own intra-multicast bandwidth sharing strategy which may implement either a multicast pricing policy [8] or an intra-multicast bandwidth sharing strategy [10].

To implement our approach, we propose to use a two classes CBQ/WRR-like scheduler [6]: one for the unicast flows and the other one for multicast flows. We call our scheduler Service-Based Queuing (SBQ) because it distinguishes between two different transfer services: unicast and multicast services. SBQ integrates a method to dynamically vary the weights of the queues in order to match the expected bandwidth share between unicast and multicast flows. Upon a packet arriving, the router has to classify and redirect it to the appropriate queue.

We use simulation to evaluate the effectiveness and performance of our scheme for various sources including not only TCP, UDP, and multicast CBR sources, but also multicast sources that implement the recently proposed layered multicast congestion control FLID-DL [13]. Simulations are done for very heterogeneous network and link characteristics and with different starting time, finish time, packet size, rate, and number of receivers in the multicast sessions.

The body of the paper is organized as follows. Section 2 details the inter-service fairness notion. We explore and discuss our scheduler in Section 3. In Section 4, the simulation results will be presented. Section 5 concludes this paper by summarizing our findings and outlining future work.

2 Inter-service Fairness

In the informational IETF standard [14], the authors recommend that each end-to-end multicast congestion control should ensure that, for **each** source-receiver pair, the multicast flow must be TCP-friendly. We believe that this recommendation has been done because there is no network support to guarantee the tcp-friendliness and that it was an anticipated requirement which aims to encourage the fast deployment of multicast in the Internet.

We propose a new notion of the unicast and multicast fairness called: the *inter-service fairness* which is defined as follows.

Definition 1: ***The Inter-service fairness:*** *The multicast flows must remain* ***globally*** *TCP-friendly and not TCP-friendly for individual flows. In other words, we should ensure that the* ***sum of multicast flows rate*** *does not exceed the* ***sum of their TCP-friendly rate****.*

The TCP throughput rate R_{TCP}, in units of **packets per second**, can be approximated by the formula in [7]:

$$R_{TCP} = \frac{1}{RTT\sqrt{q}(\sqrt{\frac{2}{3}} + 6\sqrt{\frac{3}{2}}q(1+32q^2))} \tag{1}$$

where R_{TCP} is a function of the packet loss rate q, the TCP round trip time RTT, and the round trip time out value RTO, where we have set $RTO = 4RTT$ according to [16]. Since the multicast analogue of RTT is not well defined, a target value of RTT can be fixed in advance to generate a target rate R_{TCP}.

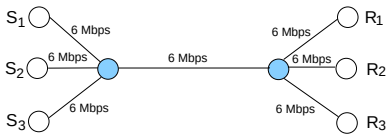


Fig. 1. A topology used as example for the inter-service fairness definition. All links have the same Round Trip Time (RTT).

We illustrate the inter-service fairness definition using the topology shown in Figure 1. We consider two multicast sessions, one from source S_1 sending to R_1 and the other one from source S_2 sending to R_2 and one unicast session from S_3 sending to R_3. Let r_1, r_2, and r_3, be the *inter-service fair share* of session S_1, S_2, and S_3, respectively. The tcp-friendly rate is equal to $\frac{6}{3} = 2$ Mbps given that all links have the same RTT. When applying the inter-service fairness definition to share the bandwidth between the three sessions, S_1 and S_2 will get together an aggregated fair share equal to 4 Mbps and session S_3 will get alone a fair share r_3 equal to 2 Mbps. Our definition of the inter-service fairness does not specify the way how the bandwidth should be shared among multicast competing flows. Therefore, each vector (r_1,r_2) where $r_1 + r_2 = 4$ Mbps is considered as a feasible intra-multicast fair share solution.

3 The SBQ Scheduler

3.1 Principals and Architecture

In order to implement the fluid model algorithm that integrates our inter-service fairness definition given in Section 2, we propose to use a CBQ-like [6] scheduler with two queues, one for each class as shown in Figure 2. We call our scheduler SBQ (Service-Based Queuing) because it differentiates between the packets according to their transfer service: unicast or multicast.

Before being queued, unicast and multicast packets are classified into two separated classes. We use the Weighted Round Robin (WRR) algorithm which is the most widely implemented scheduling algorithm as of date [6]. This is due to its low level of complexity and its ease of implementation which follows hence. In this scheduler, packets receive service according to the priority queue to which they belong. Each queue has an associated weight. In every round of service, the number of packets served from a queue is proportional to its associated weight and the mean packet size.

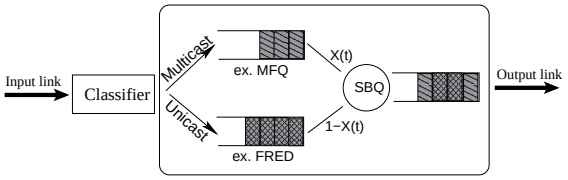


Fig. 2. SBQ scheduler architecture

As buffer management, we use our recently proposed MFQ [3] mechanism for the multicast queue and another queue management mechanism (RED, FRED, WRED, etc.) for the unicast queue.

Our scheduler operation is illustrated by the following example: consider a WRR server that uses two queues: U (for Unicast) and M (for Multicast) with weights 0.6 and 0.4, respectively. Let the mean packet size for U and M be 1000 Bytes and 500 Bytes, respectively. The weights are first normalized by the mean packet size to get 0.6 for class U and 0.8 for class M. The normalized weights are then multiplied by a constant to get the lowest possible integer. In this case we get 6 and 8 for U and M, respectively. This means that in every round of service 6 packets are served from queue U and 8 packets are served from queue M.

3.2 Scheduler Configuration

When configuring WRR to share the link bandwidth between the two classes. At time t we configure the multicast class (M queue) with a weight equal to $X(t)$ and the unicast class (U queue) with a weight equal to $(1 - X(t))$.

To implement our inter-service fairness notion defined in Section 2 and to be as close as possible to the fluid model algorithm developed in Section 3, we propose to update the weight $X(t)$ at time t as follows:

$$X(t) = \min\left(\frac{\sum_{i=1}^{i=m(t)} R_{TCP_i} * S_i}{C}, \frac{m(t)}{u(t) + m(t)}\right) \tag{2}$$

where:

- R_{TCP_i} is the TCP-friendly throughput rate of the multicast flow i estimated using Eq. 1,
- S_i is the average packet size in bytes,
- C is the link capacity in **bytes per second**,
- $u(t)$ is the number of active unicast flows at time t,
- $m(t)$ is the number of active multicast flows at time t.

To be **globally fair** against TCP-connections, the sum of the rates of $m(t)$ active multicast flows must not exceed the sum of that of their $m(t)$ single TCP flows over large time scales. This corresponds to the first term of Eq. 2. The second term of this equation allocates instantaneous bandwidth fairly between unicast

and multicast flows by sharing it proportionally to the number of flows in the two queues. Using this simplistic and efficient formula of $X(t)$, we can guarantee both short-term and long-term fairness. Indeed, the two terms used in $X(t)$ configuration allow us to ensure both short-term max-min fairness between active flows based on a local information about the number of flows and a long-term tcp-friendliness between active sessions based on a global information concerning the rates of multicast sources. The tcp-friendliness term (the first one) is useful when there is another bottleneck in the multicast delivery tree.

A possibility way to extend the configuration of $X(t)$ is to add a third term referring to the maximum portion of the link capacity that should not be exceeded by multicast flows. This value can be tuned by the ISP depending on a chosen policy used to handle multicast connections crossing its network.

Upon each change on the number of active flows in the unicast or multicast queue, the weights of both queues are updated to match the new fairness values. Both queues priority are set to 1.

To compute the value of $X(t)$, each SBQ router has to know the TCP-friendly rate of active multicast flows and maintain only the aggregated rate to be used in the first term of Eq. 2.

The TCP-friendly rate of a multicast session corresponds to the sending rate of the source. In single rate multicast transmission [18], every receiver periodically estimates its TCP-friendly reception rate using for example the formula 1 and reports this rate to the source. The source determines the lowest rate among all receivers rates and use it to send data downstream to all receivers. In the other hand, in multi-rate multicast transmission [12], the source sends data using several layers. For each layer, the source uses a specific sending rate depending on the data encoding scheme. The receivers join and leave the layers according to their reception TCP-friendly reception rates computed using Eq. 1.

For both cases, the source tcp-friendly throughput rate can be included in the IP packet header by the multicast source or the source's Designed Router. This technique is largely used by many other mechanisms such as CSFQ [15] for different purposes. Thus, a SBQ intermediate router gets the rates from the IP multicast packet headers and computes their aggregated value according to Eq. 2.

3.3 Weights Updating Time

In this sub-section, we try to answer the question: How often we update the weights ? In other words, what is the time-scale on which we should look at the bandwidth allocation. There is a tradeoff between complexity and efficiency when choosing the time-scale value. Indeed, larger time-scale are not suitable for short-lived TCP connections which are the most of TCP connections currently in the Internet[1]. In the other hand, what happiness on shorter time-scales if we consider fairness on longer time-scale. We do not claim that there is an optimal

[1] Internet traffic archive: `http://www.cs.columbia.edu/~hgs/internet/traffic.html`

value of the time-scale which can be applied for each type of traffic and which leads to both less complexity and good efficiency.

The updating time is designed to allow the update of weights to take effect before the value is updated again, it should be longer than the average round trip time (RTT) among the connections passing through the router. In the current Internet, it can be set in the range from 10 ms up to 1 sec, so this property can guarantee that the unfairness can't increase very quickly and make the queue parameters stable. But this parameter can't be set too big so that the scheduler weight can't be adaptive enough to the network dynamics.

3.4 Counting Unicast Connections

To update the value of the weight used in our scheduler computed using Eq. 2, we need to know the number of multicast and unicast flows. While the former is provided by MFQ, the latter could be obtained through the use of a flow-based unicast active queue management mechanism such as FRED [11]. However, unicast flow-based AQM are not available everywhere and most of current routers use a FIFO or RED [5] schemes which don't provide the number of unicast connections. That's why, we propose hereafter a simple method which we use to estimate the number of unicast connections in the unicast queue.

SBQ counts active unicast connections as follows. It maintains a bit vector called v of fixed length. When a packet arrives, SBQ hashes the packets connection identifiers (IP addresses and port numbers) and sets the corresponding bit in v. SBQ clears randomly chosen bits from v at a rate calculated to clear all of every few seconds. The count of set bits in v approximates the number of connections active in the last few seconds. The SBQ simulations in Section 4 use a 5000-bit v and a clearing interval (t_{clear}) of 10 seconds. The code in Algorithm given in [4] may under-estimate the number of connections due to hash collisions. This error will be small if v has significantly more bits than there are connections. The size the error can also be predicted and corrected; see [4]. This method of counting connections has two good qualities. First, it requires no explicit cooperation from unicast flows. Second, requires very little state: on the order of one bit per connection.

3.5 Deployment Issues

The deployment of CBQ/WRR-like algorithms in the Internet may raise some open questions for large deployment. The scalability issue is the main barrier of their large deployment in the Internet. We mean by the scalability, the ability of the mechanism to process a very large number of flows with different characteristics at the same time. We believe that our scheduler can be deployed in large networks thanks to two mainly key points: (1) it uses only two queues, so we need to classify only two types of service: unicast and multicast flows. This task has already been done in part by the routing lookup module before the packet being queued, and (2) all unicast flows are queued in the same queue.

It is important to note that we usually associate the flow-based mechanisms support with complexity and scalability problems since they require connection specific information. These concerns are justifiable only in point-to-point connections, for which routing tables do not maintain connection-specific state. In multicasting, routing tables keep connection specific state in routers anyway; namely, the multicast group address refers to a connection. Thus, adding multicast flow specific information is straightforward and increases the routing state only by a fraction.

Comparing to CBQ/WRR, our mechanism is less complex to be deployed in the Internet. It should be noted that even CBQ is now supported by large number of routers and many research works such as [17] demonstrate its deployment feasibility.

One major advantage of our approach is that it minimizes the complexity of designing multicast congestion control. Indeed, the network guarantees that the multicast flows will share fairly the bandwidth with competing unicast flows. Moreover, our scheme provides to the ISPs a flexible way to define and implement their own intra-multicast fairness strategy. The simulation results presented in the next section will confirm our claims.

4 Simulation Methodology and Results

We have examined the behavior of our scheme under a variety of conditions. We use an assortment of traffic sources and topologies. All simulations were performed in network simulator (ns-2).

4.1 Single Bottleneck Link

We validate our scheme for the topology of Figure 3 which consists of a single congested link connecting two routers n_1 and n_2 and having a capacity C equal to 1 Mbps and a propagation delay D equal to 1 ms.

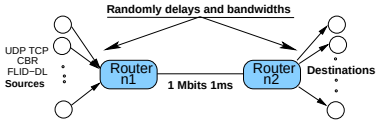


Fig. 3. A single congested link simulation topology

We configure our scheduler in the bottleneck link from router n_1 to router n_2. The maximum buffer size $qlim$ of both queues is set to 64 packets. Other links are tail-drop and they have sufficient bandwidth to avoid packets loss. We assume 32 multicast sources and 32 unicast sources that compete to share the link capacity. We index the flows from 1 to 64. The 32 multicast sources are divided as follows:

- Flows from 1 to 16: CBR sources. These sources implement no type of congestion control mechanism (neither application-level nor transport-level).
- Flows from 17 to 32: FLID-DL (Fair Layered Increase-Decrease with Dynamic Layering) [13] sources. The FLID-DL simulation parameters are the same as those recommended in [13][2].

Each source uses 17 layers encoding, each layer is modeled by a UDP traffic source.

As outlined earlier, we use MFQ [3] as the active queue management in the multicast queue. Without loss of generality we utilize a receiver-dependent logarithm policy (the LogRD policy) proposed in [10] to share the bandwidth between multicast flows. This policy consists in giving to the multicast flow number i a bandwidth fraction equal to $\frac{1+\log n_i}{\sum_j (1+\log n_j)}$, where n_j is the number of receivers of flow j.

The 32 unicast sources are composed as follows:

- Flows from 33 to 48: UDP sources.
- Flows from 49 to 64: TCP sources. Our TCP connections use the standard TCP Reno implementation provided with ns.

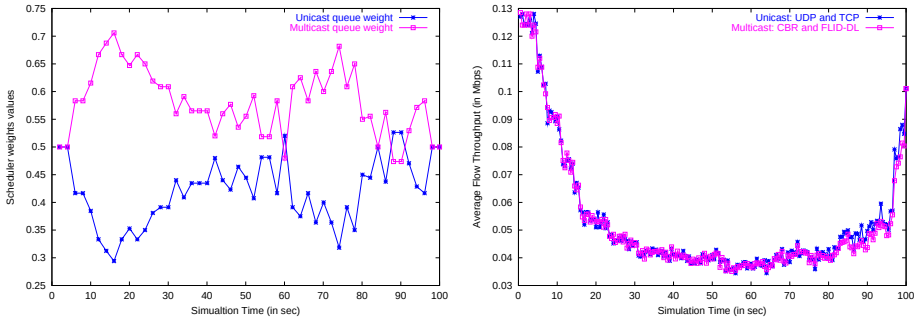
Unless otherwise specified, each simulation lasts 100 seconds and the weight updating period is set to 2 sec. Other parameters are chosen as follows:

- packet size: the packet size of each flow is randomly generated between 500 and 1000 bytes.
- starting time: the starting time of each flow is randomly generated between 0 and 20 seconds before the end of the simulation.
- finish time: the finish time of each flow is randomly generated between 0 and five seconds before the end of the simulation.
- rate: the rate of both unicast and multicast UDP flows is randomly generated between 10 Kbps and 100 Kbps.
- number of receivers: the number of downstream receivers of the 32 multicast sessions is randomly generated between 1 and 64.

The four first unicast UDP and multicast UDP flows are kept along the simulation time (start time = 0 sec, and finish time = 100 sec) to be sure that the link will be always congested. Each one of these flows is sending at a rate equal to $\frac{1\ Mbps}{8} = 125\ Kbps$. Initially (at $t = 0$ sec), the weight X is set to 0.5.

In Figure 4(a), we plot the variation of unicast and multicast queue weights in function of the simulation time. As we can easily see the value of weights change during the simulation because they depend on the number of active unicast and multicast flows in the queues. The multicast weight increases when the unicast weight decreases and vice versa.

[2] These parameters are used in the ns implementation of FLID available at `http://dfountain.com/technology/library/flid/`.



(a) Scheduler weights variation in function of the simulation time

(b) The variation of the unicast and multicast average rate in function of the simulation time over 500 msec

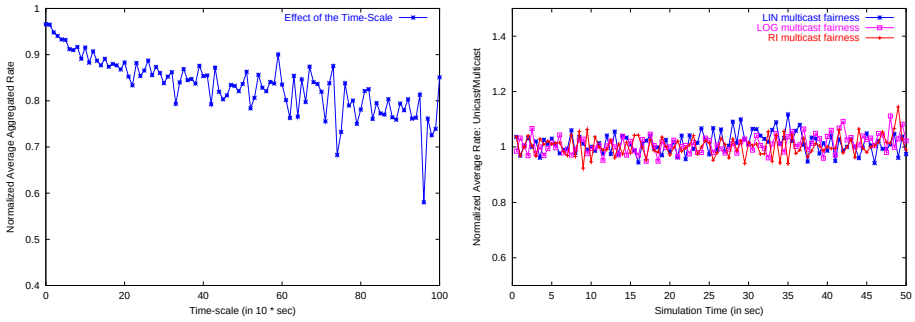
Fig. 4. Weights and average rates variation

We now interest in the inter-service fairness defined in Section 2 which is the main performance metric of our scheme. To this end, we compare the average unicast an multicast rates over 500 *msec* of simulations. We show in Figure 4(b), the variation of the unicast and the multicast average rate in function of the simulation time.

As we can observe the results match exactly what we expect. Two main observations can be done from the plots of Figure 4(b). Firstly, there is a fluctuation on the average aggregated rate for unicast and multicast flows. Secondly, the multicast average rate is very close to the unicast average rate. This demonstrates the ability of SBQ to share the bandwidth fairly between unicast and multicast flows according to our inter-service fairness notion defined in Section 2.

In order to evaluate the impact of the time-scale value on the performance of SBQ, we measure the normalized aggregated rate (average unicast rate/average multicast rate) obtained for various values of the time-scale. In Figure 5(a), we show the variation of this metric in function of the time-scale value. We can conclude that using a 1 sec time-scale, we can reach 93 % of the performance of SBQ. In other words, the use of an updating period equal to 1 sec leads to a good tradeoff between complexity and efficiency.

We claimed earlier that our scheduler is flexible in the sense that its performance is independent of the intra-multicast fairness function. To argument this affirmation, we compare in Figure 5(b) the normalized rate over 500 *msec* of simulation time for linear (LIN), logarithm (LOG), and receiver-independent



(a) Sensitivity of SBQ performance on the time-scale value

(b) The normalized rate when modifying the intra-multicast fairness function over 500 msec

Fig. 5. Sensitivity of SBQ performance on time-scale value and fairness function

(RI) bandwidth sharing policies which are defined in [10][3]. As we can see the normalized average rate is always varying around 1 (between 0.82 and 1.18) for the three cases.

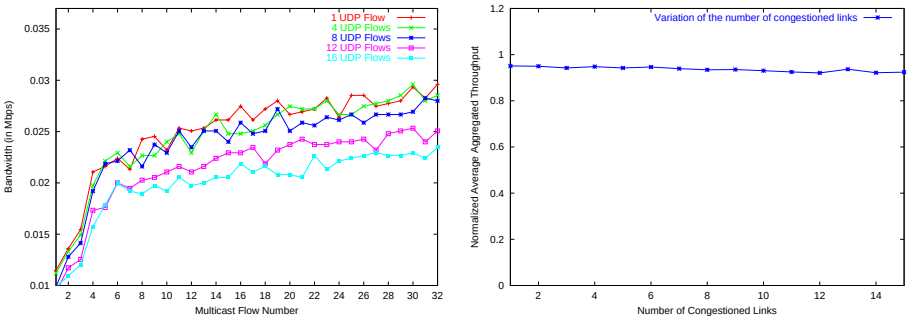
We interest in the bandwidth shared between multicast flows obtained by MFQ buffer management mechanism. The multicast flow number i is assumed to have exactly i downstream receivers (members). We use a logarithm multicast bandwidth allocation scheme. We vary the number of UDP unicast flows from 1 to 16 flows. In Figure 6(a), we show the bandwidth fair rate obtained for the 32 multicast flows. It is very clear that the shape of flows rate curves follow a logarithm function of the number of downstream receivers.

4.2 Multiple Bottleneck Links

In this sub-section, we analyze how the throughput of multicast and unicast flows is affected when the flow traverses L congested links. We performed two experiments based on the topology of Figure 7. We index the links from 1 to L and the capacity C_1 of the link number 1 is set to 1 Mbps.

We use the same source as the case of single bottleneck and we measure the aggregated bandwidth received by each service type: unicast and multicast type in function of the number of congested links. A link j, $2 \leq j \leq 10$, is kept congested by setting its capacity C_j to $C_{j-1} - 50$ Kbps.

[3] Assume that we have n active multicast flows and denote by n_i the number of downstream receivers of flow i. The RI, LIN, LOG bandwidth sharing policies consist to give to flow i a bandwidth share equal to $\frac{1}{n}$, $\frac{n_i}{\sum_j n_j)}$, and $\frac{1+\log n_i}{\sum_j (1+\log n_j)}$, respectively.



(a) Rates of multicast flows when using a logarithm multicast bandwidth sharing function

(b) The normalized rate as a function of the number of congestioned links

Fig. 6. SBQ Performance for multiple bottleneck links

We plot in Figure 6(b), the variation of the normalized average rate as a function of the number of congested links. As we can see, the normalized average rate remains close to 1 even when the number of congested links increases.

5 Conclusion and Future Work

In this paper, we have presented a CBQ-like scheduler for bandwidth sharing between unicast and multicast flows. The scheduler uses two queues: one for unicast and the other one for multicast. We used a simplistic and efficient dynamic configuration method of the WRR scheduler to achieve the expected sharing based on a new fairness notion called *the inter-service fairness*.

The buffer management mechanism used in the multicast queue was MFQ, a new scheme that we have proposed in [3] and which provides the expected multicast bandwidth sharing between multicast flows using a single FIFO queue.

To validate our scheme, we simulated a very heterogeneous environment with different types of sources, starting times, sending rates, delays, and packets sizes. We demonstrated that SBQ achieves the expected results in the sense that the bandwidth is shared fairly between unicast and multicast flows according to our new definition of fairness.

Future work could evaluate the performance for other types of multicast traffic that include others application-based or transport-based congestion control mechanisms. Another area of future study is to develop a new multicast congestion control mechanism that uses a small but efficient help from our scheduler.

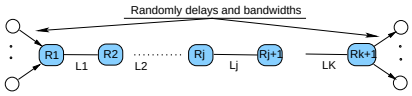


Fig. 7. Topology for analyzing the effects of multiple congested links

References

1. J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, *A digital fountain approach to reliable distribution of bulk data transfer,* In Proc. of ACM SIGCOMM'98, September 1998.
2. D. M. Chiu., *Some Observations on Fairness of Bandwidth Sharing*, In Proc. of ISCC'00, July 2000.
3. F. Filali and W. Dabbous, *A Simple and Scalable Fair Bandwidth Sharing Mechanism for Multicast Flows*, In Proc. of IEEE ICNP'02, November 2002.
4. F. Filali and W. Dabbous, *SBQ: Service-Based Queuing*, INRIA Research Report, March 2002.
5. S. Floyd, V. Jacobson, and V. Random, *Early Detection gateways for Congestion Avoidance,* IEEE/ACM TON, V.1 No.4, pp. 397-413, August 1993.
6. S. Floyd and V. Jacobson, *Link-sharing and Resource Management Models for Packet Networks*, IEEE/ACM TON, V. 3, No.4, 1995.
7. S. Floyd, M. Handley, J. Padye, and J. Widmer, *Equation-based congestion control for unicast applications*, In Proc. of ACM SIGCOMM'00, August 2000.
8. T. N.H. Henderson and S. N. Bhatti, *Protocol-independent multicast pricing*, In Proc of NOSSDAV'00, June 2000.
9. F. Kelly, A. Maulloo and D. Tan, *Rate control in communication networks: shadow prices, proportional fairness and stability*, Journal of the Operational Research Society 49, pp. 237-252, 1998.
10. A. Legout, J. Nonnenmacher, and E. W. Biersack, *Bandwidth Allocation Policies for Unicast and Multicast Flows,* IEEE/ACM TON, V.9 No.4, August 2001.
11. D. Lin and R. Morris, *Dynamics of Random Early Detection*, In Proc. of ACM SIGCOMM'97, September 1997.
12. M. Luby, V. Goyal, and S. Skaria, *Wave and Equation Based Rate Control: A massively scalable receiver driven congestion control protocol*, draft-ietf-rmt-bb-webrc-00.txt, October 2001.
13. M. Luby, L. Vicisano, and A. Haken, *Reliable Multicast Transport Building Block: Layered Congestion Control*, Internet draft: draft-ietf-rmt-bb-lcc-00.txt, November 2000.
14. A Mankin, et al., *IETF Criteria for evaluating Reliable Multicast Transport and Applications Protocols*, IETF RFC 2357, June 1998.
15. I. Stoica, S. Shenker, and H. Zhang, *Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks*, In Proc. of SIGCOMM'98.
16. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, *Modeling TCP throughput: a simple model and its empirical validation*, In Proc. of ACM SIGCOMM'98.
17. D. C. Stephens, J.C.R. Bennet, and H. Zhang, *Implementing scheduling algorithms in high speed networks*, IEEE JSAC, V. 17, No. 6, pp. 1145-1159, June 1999.
18. J. Widmer and M. Handley, *TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification*, draft-ietf-rmt-bb-tfmcc-00.txt, November 2001.