Fair Bandwidth Sharing Between Unicast and Multicast Flows in Best-Effort Networks

Fethi Filali and Walid Dabbous
INRIA, 2004 Route des Lucioles, BP-93,
06902 Sophia-Antipolis Cedex, France

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 also describe a new active queue management mechanism called MFQ (Multicast Fair Queuing) to fairly share the allowed multicast bandwidth among competing flows in the multicast queue.

The simulation results obtained for very heterogeneous sources and links characteristics suggest that, on the one hand, SBQ achieves the expected aggregated bandwidth sharing among unicast and multicast service, and on the other hand the multicast flows remain TCP-friendly.
Keywords: Multicast fairness, Fair bandwidth sharing, Two classes scheduler mechanism, Active queue management, Membership Information .

1 Introduction

It is widely accepted that one of the several factors inhibiting the usage of the IP multicast is the lack of good and 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 number of layers at different rates.

In this paper, we develop a novel approach that helps multicast congestion control mechanisms to fairly exist with TCP protocol. Our approach is based on a new fairness notion, *the inter-service fairness*, which is used to share the bandwidth fairly between unicast and multicast services. Our fairness definition requires that the aggregated multicast traffic remains *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 the ISPs to define their own intra-multicast bandwidth sharing strategy which may implement either a multicast pricing policy [15] or an intra-multicast bandwidth allocation strategy [18].

To implement the inter-service fairness notion, we propose a two classes CBQ/WRR-like scheduler [13]: 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 two queues in order to match the expected bandwidth share between unicast and multicast flows. Upon a packet arrival, the router has to classify and redirect it to the appropriate queue.

The aim of SBQ in the global architecture is to achieve the bandwidth sharing between unicast and multicast classes. In order to share the multicast bandwidth fairly among all competing multicast flows in the multicast queue, we propose and evaluate a new active queue management mechanism for multicast

flows called MFQ (Multicast Fair Queuing) which represents another key component. MFQ uses a single FIFO queue to share the bandwidth according to a pre-configured bandwidth allocation scheme. It also implements a novel bandwidth sharing notion, called Multicast Allocation Layer (MAL). Based on this notion, MFQ classifies multicast packets into layers and adjusts their weights in order to provide a bandwidth sharing being as close as possible to that given by the fluid model algorithm. Since social, economic, and technical issues lead the ISPs to implement different fairness policies, considering their business strategy, we made the choice that MFQ mechanism will be independent of the fairness function used. Indeed, the multicast bandwidth allocation module may implement either a multicast fairness function as those described in [18] or a multicast pricing model [15].

To the best of our knowledge there is no prior work on unicast and multicast bandwidth sharing using a scheduling mechanism in the open literature. Furthermore, we consider our scheduler a promising avenue in developing congestion control mechanisms for multicast applications, and so an additional motivation for this work is to lay a sound basis for further development of multicast congestion control with a small but efficient help from the network. Note that our proposals described in this paper concern only best-effort networks and their deployment in DiffServ-like networks are described in [12].

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 scheme FLID-DL [21]. Simulations are done for heterogeneous network and link characteristics and with different starting time, finish time, packet size, rate, and number of receivers in the multicast sessions.

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.

The body of this paper is organized as follows. In Section 2, we present the inter-service fairness notion. The fluid model algorithm is presented in Section 3. Section 4 details the characteristics of the SBQ scheduler and the MFQ buffer management mechanism is described in Section 5. We analyze in Section 6 the implementation and deployment complexity of our schemes. The performance evaluation of SBQ in best-effort networks is explored in Section 7. Section 8 summarizes our main findings and outlines future research directions.

2 Inter-Service Fairness

In the informational IETF standard [22], the authors recommended 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. In addition, the multicast congestion control mechanisms that tried to achieve the TCP-fairness criterion are not always fair with TCP [5, 27] especially under variable network conditions.

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

Definition 2.1 *The Inter-service fairness: The multicast flows must remain globally TCP-friendly and not individually TCP-friendly for each flow. In other words, we should ensure that the sum of multicast flows rate does not exceed the sum of their TCP-friendly rates.*

The TCP throughput rate R_{TCP}, in units of **packets per second**, can be approximated by the formula in [14]:

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

where R_{TCP} is a function of the packet loss rate q, the TCP round trip time RTT, and the retransmission timeout value RTO, where we have set $RTO = 4RTT$ according to [24]. 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}.

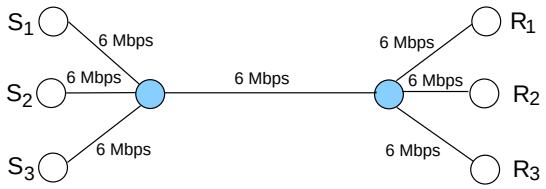


Figure 1: A topology used to illustrate the inter-service fairness definition. All links have the same Round Trip Time (RTT).

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 source S_3 sending to R_3. Let r_1, r_2, and r_3, be the *inter-service fair share* of source 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 sources, S_1 and S_2 will get together an aggregated fair share equal to 4 Mbps and source 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 Fluid Model Algorithm

We consider a network as a set of links $\Im$ where each link l_j has a capacity $C_j > 0$. In addition to unicast flows, we a known number of multicast flows with different number of receivers. All flows (unicast and multicast) compete for access to communication links. A multicast session S_i is a tuple $(X_i, \{r_{i,1}, ..., r_{i,k}\})$ of session members: X_i is the session sender that transmits data within a network; each $r_{i,p}$ is a receiver that receives data from X_i. Each session contains exactly one sender and at least one receiver. We denote G_j the multicast group number j to be a multicast group to which belong one or more multicast sessions. We write $r_{i,p} \in S_i$ to indicate that receiver $r_{i,p}$ is a member of the multicast session S_i and $S_i \in G_j$ to indicate that the multicast session S_i belongs to group G_j.

We define $N_{i,j}$ to be the number of receivers in session S_i whose path toward the source includes link l_j and define N_j to be the number of all receivers whose multicast delivery tree includes link l_j, i.e., $N_j = \sum_i N_{i,j}$.

We now consider a single link with a capacity equal to C. We assume n active TCP flows and m active multicast flows arriving to the link. The TCP source number i sends at the instantaneous rate $\alpha_i(t)$ and the multicast source number i sends at the instantaneous rate $\beta_i(t)$, in bits per second.

Unicast max-min fair bandwidth allocations [3] are characterized by the fact that all TCP flows that are bottlenecked (i.e., have packets dropped) by a router have the same output rate. We call this rate the *unicast allocation rate* of the server; let $\lambda(t)$ be the unicast allocation rate at time t. In general, if max-min bandwidth allocations are achieved, each unicast flow i receives service at a rate given by $min(\alpha_i(t), \lambda(t))$. For multicast flows, the nature of the expected allocation is not yet well defined. It can be either determined by a fairness function scheme depending on the group membership [18] or by a multicast pricing model [15]. To be independent of the allocation strategy used, we define a vector $\gamma(t) = (\gamma_1(t), \gamma_2(t), ..., \gamma_m(t))$ giving the expected allocation at the instantaneous time t. As explained in Section 5.1, the value of $\gamma_i(t)$ may be either a set value or a function of the number of competing multicast groups, the number of flows per group, and the number of receivers per flow. We call this vector of fair, the *multicast allocation vector*. If the fair share is achieved, the multicast flow number i receives service at a rate given by $min(\beta_i(t), \gamma_i(t))$.

Let $A(t)$ be the total arrival rate: $A(t) = \sum_{i=1}^{i=n} \alpha_i(t) + \sum_{i=1}^{i=m} \beta_i(t)$. If $A(t) > C$ the congestion phenomena holds and then the unicast allocation rate $\lambda(t)$ and the multicast allocation vector $\gamma(t)$ which corresponds to the inter-service fairness definition given in Section 2 are the unique solution to

$$\sum_{i=1}^{i=n} min(\alpha_i(t),\ \lambda(t)) + \sum_{i=1}^{i=m} min(\beta_i(t),\ \gamma_i(t)) = C \quad (2)$$

subject to:

$$\sum_{i=1}^{i=m} \gamma_i(t) \leq m\lambda(t) \quad (3)$$

This constraint is added in order to guarantee that the aggregated multicast bandwidth share rate does not exceed that of m equivalent unicast flows. In other words, each multicast flow will get an average rate equal to $\lambda(t)$.

If $\alpha_i(t) > \lambda(t)$, then the fraction of bits $\frac{\alpha_i(t)-\lambda(t)}{\alpha_i(t)}$ will be dropped, and the unicast flow i will have an output rate of exactly $\lambda(t)$. The arrival rate to the next hop is given by $min(\alpha_i(t),\ \lambda(t))$.

If $\beta_i(t) > \gamma_i(t)$, then the fraction of bits $\frac{\beta_i(t)-\gamma_i(t)}{\beta_i(t)}$ will be dropped, and the multicast flow i will have an output rate of exactly $\gamma_i(t)$. The arrival rate to the next hop is given by $min(\beta_i(t),\ \gamma_i(t))$.

As mentioned above, the multicast allocation vector can be computed using a multicast fairness function that depends on the number of receivers which are distributed in the multicast delivery tree. Therefore, the number of downstream receivers in each router of the multicast delivery tree does not remain constant because some receivers may be reached via different interfaces. Thus, $\gamma(t)$ may be different in the tree branches even when there is no more competing multicast flow. To illustrate this, we consider two successive routers r_1 and r_2 composing a branch of the multicast tree of flow i and denote by R_1 and R_2 the number of receivers downstream to r_1 and r_2, respectively. One can easily write the following inequality $R_2 \leq R_1$ because router r_1 is close to the multicast source than router r_2. Hence, an immediate inequality between the flow weights in the two routers holds: $\gamma_i^2(t) \leq \gamma_i^1(t)$ which means that the flow weight in router r_2 ($\gamma_i^2(t)$) is less than or equal to that in router r_1 ($\gamma_i^1(t)$).

4 The Service-Based Queuing (SBQ) Scheduler

4.1 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 CBR/WRR-like scheduler [13] but with only 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.

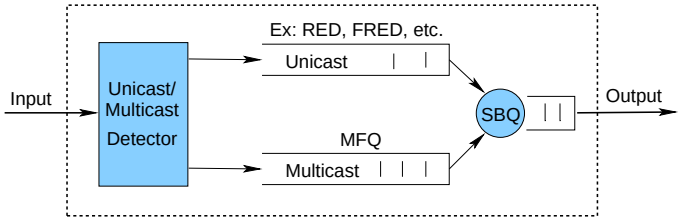


Figure 2: SBQ scheduler architecture

Before being queued, unicast and multicast packets are classified into two separated classes. We use a modified version of the Weighted Round Robin (WRR) algorithm which is the most widely implemented scheduling algorithm as of date [13]. 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 service quantum[1] of

[1]The service quantum of a given flow is expressed in term of packets belonging to this flow which could be served in a service round.

fbws 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.

As buffer management schemes: we use MFQ (Multicast Fair Queuing) mechanism (which will be detailed in Section 5) for the multicast queue and another queue management mechanism (RED, FRED, WRED, etc.) for the unicast queue.

The SBQ scheduler's 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.

4.2 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 ensure that the bandwidth sharing is 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:

$$X(t) = \min\left(\frac{\sum_{i=1}^{i=m(t)} R_{TCP_i} * S_i}{C}, \frac{m(t)}{u(t) + m(t)}\right) \tag{4}$$

where:

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

To be **globally fair** against TCP-connections, the sum of the rates of $m(t)$ active multicast fbws must not exceed the sum of that of their $m(t)$ single TCP fbws rate over large time scales. This corresponds to the first term of Eq. 4. The second term of this equation allocates instantaneous bandwidth fairly between unicast and multicast fbws by sharing it proportionally to the number of fbws in the two queues. We believe that 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 fbws based on a local information about the number of fbws and a long-term TCP-friendliness between active sessions based on a global information concerning the rates of multicast sources.

A possible 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 fbws. This value can be tuned by the ISP depending on a chosen policy used to handle multicast connections crossing its network. In this case, the configuration of the weight $X(t)$ of the multicast queue will be done as follows:

$$X(t) = \min\left(\max\left(\frac{\sum_{i=1}^{i=m(t)} R_{TCP_i} * S_i}{C}, minBDW_{mcast}\right), \frac{m(t)}{u(t) + m(t)}\right), \tag{5}$$

where $minBDW_{mcast}$ is the minimum capacity fraction that should be given to multicast fbws [2].

[2] The weight of the unicast queue is therefore equal to $1 - X(t)$.

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. 4. The TCP-friendly rate of a multicast session corresponds to the sending rate of the source. In single rate multicast transmission protocols, such as TFMCC [28], 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 uses it to send data downstream to all receivers. In multi-rate multicast transmission such as RLC [27], WEBRC [20] and FLID-DL [5], 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 Designated Router (DR). This technique is largely used by many other mechanisms such as CSFQ [26], TUF [8] 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. 4 or Eq. 5.

4.3 Weights updating time

In this sub-section, we 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[3]. On the other hand, what happens on shorter time-scales if we consider fairness on longer time-scale. We do not claim that there is an optimal 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 time-scale is designed to allow the update of weights to take effect before their values are updated again. It should be therefore 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.

We will show in the simulation section that the use of a time-scale equal to 1 sec, which we believe an accepted value, can provide a good tradeoff between efficiency and complexity.

4.4 Counting unicast connections

To update the value of the weight used in our scheduler computed using Eq. 4, 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 [19]. However, unicast flow-based AQM are not available everywhere and most of current routers use a FIFO or RED [12] 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. Appendix B contains pseudo-code for this algorithm. The SBQ simulations in Section 7 use a 5000-bit v and a clearing interval (t_{clear}) of 10 seconds The code in Appendix B 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. This method of counting connections has two good qualities. First, it requires

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

no explicit cooperation from unicast flows. Second, requires very little state: on the order of one bit per connection.

5 The Multicast Fair Queuing (MFQ) Mechanism

Two modules compose our mechanism:

- The multicast bandwidth allocation module which computes the expected fair share for each active multicast flow [4].
- The buffer management module which uses a single FIFO queue and interacts with the first module to decide the drop preference in order to achieve the expected bandwidth sharing.

In Figure 3, we show a simplified architecture of MFQ. In the following sections, we explore and discuss the two modules.

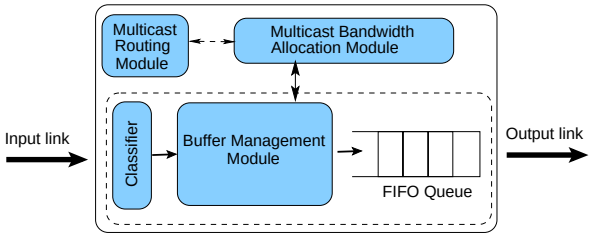


Figure 3: A simplified MFQ architecture

5.1 Multicast Bandwidth Allocation Module

We first present a general framework for the multicast fairness notion. Then, we enumerate some inter-multicast fairness candidates functions that could be implemented by the ISPs. We consider the two existing multicast service models: the ASM (Any Source Multicast) [10] model and the SSM (Single Source Multicast) model [16].

5.1.1 General Framework

The multicast bandwidth allocation module determines the link capacity fraction[5] that should be allocated to the flow to which the incoming packet belongs. We develop hereafter a general framework of the multicast bandwidth allocation module. As we have pointed out earlier, the multicast fairness function may depend on the number of groups, the number of flows per group, and the number of receivers per flow. We assume that each ISP has a single and clearly defined multicast bandwidth sharing policy. This policy can be configured in all or some routers inside its network.

Using the network model introduced in Section 3, we define the bandwidth γ_{ij}, in bits per second, allocated to group G_i, in link l_j as follows:

$$\gamma_{ij} = F_1(G_i) * C_j \tag{6}$$

where $F_1(.)$ is the ***inter-group multicast fairness function*** that implements the bandwidth sharing policy among active multicast groups. In other words, $F_1(G_i) * C_j$ is the maximum bandwidth, in bits per second, that should be allocated to group G_i in link l_j.

[4] A multicast flow is considered instantaneously active if it has at least one packet in the queue.

[5] Throughout this dissertation we use the terms "link capacity fraction", "flow weights", and "flow bandwidth allocation", interchangeably.

In the ASM (Any Source Multicast) service model [9], a multicast group may have one or more sessions (fbws) from different sources that share the same communication link. The bandwidth allocated to fbw k of the multicast session $S_k \in G_i$ in link l_j is given by the following expression:

$$\lambda_{kj} = F_2(S_p / S_p \in G_i) * \gamma_{ij} \tag{7}$$

where γ_{ij} is computed using Eq. 6. The function $F_2(.)$ determines the fraction of the bandwidth which has already been allocated to group G_i and that should be given to the session S_k. We call this function the ***intra-group multicast fairness function***. Both F_1 and F_2 depend on the number of downstream receivers of each multicast session that belongs to the same group. The functions $F_1(.)$ and $F_2(.)$ must satisfy the following properties:

- for each multicast group G_i: $0 < F_1(G_i) < 1$, and $0 < F_2(S_p) < 1$ for each $S_p \in G_i$;
- in each link l_j: $\sum_i F_1(G_i) = 1$, and $\sum_{pS_p \in G_i} F_2(S_p) = 1$ for each group G_i.

Our main focus in this paper is not to find the "optimal" functions $F_1(.)$ and $F_2(.)$, however we will show that MFQ can adapt itself according to the bandwidth allocation function used, the number of receivers per fbw, the number of active fbws, and the number of active multicast groups.

5.1.2 Examples of inter-multicast fairness functions

In the ASM service model [9], it is possible to have two or more different sources sending to the same multicast group G_i. Therefore, it makes sense to use the function F_1 to share the bandwidth between multicast competing groups. Let's assume that the capacity C_j of link l_j is **equitably** shared between them, then the group G_i gets a bandwidth share γ_{ij} equals to $F_1(G_i) * C_j = \frac{1}{g} * C_j$, where g is the total number of distinct groups. If F_2 is a logarithm function of the number of receivers, the session $S_k \in G_i$ will get a bandwidth share equal to $\lambda_{kj} = \frac{1+\log n_k}{\sum_{p=1}^{p=m_i}(1+\log n_p)} * \frac{1}{g} * C_j$, where m_i is the number of sessions that belong to group G_i, and n_p is the number of receivers of session S_p.

In the SSM service model [16], each sender may use a different group address and the multicast session is identified by the couple (sender address, group address). Thus, there is only one source per multicast group and there is therefore no need of using function F_1. In the case of using a logarithm bandwidth sharing function, the session S_k will get a bandwidth share equal to $\lambda_{kj} = \frac{1+\log n_k}{\sum_{p=1}^{p=n}(1+\log n_p)} * C_j$, where n is the total number of competing sessions.

5.2 Buffer Management Module

5.2.1 Module Description

The role of the buffer management module is to make the queuing/dropping decision of each incoming multicast packet. In the MFQ design phase, we have taken some key decisions in order to have a suitable mechanism independent of the network and sources characteristics and especially the variation of fbws weights[6], source behavior when a packet is lost, and sources rates.

We use a single FIFO queue with a pre-configured maximum size in packets. For each multicast active fbw, we maintain a fbw state containing:

- the number of packets belonging to this fbw which are waiting to be served,
- the current fbw weights which is provided by the bandwidth allocation module.

If the fbw is new or if there is a change on the number of receivers, we get the new fbw weight from the bandwidth allocation module. We assume that we know the number of downstream receivers for each active multicast fbw and in each router belonging to the multicast delivery tree. In [11], we emphasized on

[6] In the case of using a fairness function which depends on the number of downstream receivers, all fbws weights change when at least one receiver joins or leaves one of the multicast active sessions.

this issue and we propose an extension to the multicast service model to count the number of downstream members at the senders as well as at the intermediate routers in the multicast delivery tree.

To prevent the queue from being monopolized by high-rate or bursty multicast sources, we use a pre-configured threshold variable ***thrsh***. If the mean queue size[7] is less than *thrsh* the packet will be accepted only if the number of waiting packets belonging to the flow does not exceed the allowed number (its MAL value or its MAL value plus one more packet depending on the generated number *u* as explained above).

If the mean queue size is more than the threshold *thrsh* or the queue is full, we accept the packet only if it belongs to an inactive flow. If the queue is full, we drop the incoming packet if its flow is active, otherwise we drop randomly a packet from the queue and we queue the incoming packet. By this way we allow a new multicast flow to become active and we remove the bias against bursty sources. If the packet was accepted, we update the flow state.

5.2.2 The Multicast Allocation Layer (MAL) Scheme

In order to achieve a fine-grained queuing/dropping, we introduce a new scheme, called Multicast Allocation Layer (MAL) which is a key component of the MFQ's buffer management module. We define a MAL as follows:

Definition: *A MAL is a set of flows that may have the same or different expected allocation in term of the link capacity fraction (the bit-level fairness), but they have the same allocation in term of the maximum number of packets (the packet-level fairness) allowed to be present at the same time in the queue.*

We assume that at the time t, there are n active multicast flows in the queue and that the flow f_i has a weight equal to w_i which is provided by the bandwidth allocation module.

In the Internet, the size of packets generated by a source may be different to that of packets generated by other sessions given that this parameter depends on the nature of the session (data, audio, video, etc.) and on the codecs used to generate the media. To take into account the fact the packet sizes are different, the weights w_i are normalized according to the packet size of all competing flows. So, we define the MAL mapping function F_{MAL} as follows:

$$\begin{array}{ccc} \{f_1, f_2, ..., f_j, ..., f_n\} & \longrightarrow & \{0, 1, 2, ..., qlim\} \\ f_j & \longmapsto & F_{MAL}(f_j) = \lfloor w_j * qlim \rfloor \end{array}$$

where $qlim$ is the queue size in packets. Two flows f_i and f_j belong to the same MAL number k only if $F_{MAL}(f_i) = F_{MAL}(f_j) = k$.

Given that the number of active flows change, the flows weights and the set of flows per MAL are dynamic. A MAL which has a non-empty set of flows is considered active. We can have at most $qlim$ active MALs in a queue of a size equal to $qlim$ and in this case each MAL contains only one flow.

Let us explain the MAL scheme through the example given in Figure 4. In the x-axis, we show the distribution of 21 active flows in the different MALs. As we can see, there are 5 active MALs with various size in term of the number of active flows that contains each MAL. The flows belonging to the same MAL have different weights drawn in the y-axis by vertical arrows. For example, flows f_{20} and f_3 belong to the MAL number 4 which has 5 active flows $\{f_4, f_{20}, f_{19}, f_3, f_{16}\}$.

To do a fine grain dropping, MFQ maintains for each active MAL the identity of the flow which has the highest weight among other flows in the same MAL. As we will detail in the next section, MFQ discriminates between flows belonging to the same MAL in order to achieve the expected fair share.

One can make the observation that when the flows weights are equal, the bandwidth is equitably distributed among flows given that there is only one active MAL including all active flows. We believe that the use of the MAL scheme will be also helpful even for unicast flows in differentiated networks by providing a fine-grained dropping for unicast packets belonging to the same DiffServ class when their corresponding flows have different weights [8].

[7] We use the same method as RED (Random Early Drop) [12] to estimate the mean queue size $qlen$. The formula for calculating the average queue length $qlen$ is $qlen = (1 - W_q) * qlen + W_q$, $0 \leq W_q \leq 1$. The weighted moving average formula, with weight W_q is used to filter out transient congestion. The value of W_q is set to 0.002 in all simulations.

[8] The details about how to use the MAL scheme in DiffServ networks for unicast flows is, of course, out of the scope of this work on multicast bandwidth sharing and it will be investigated in future works.

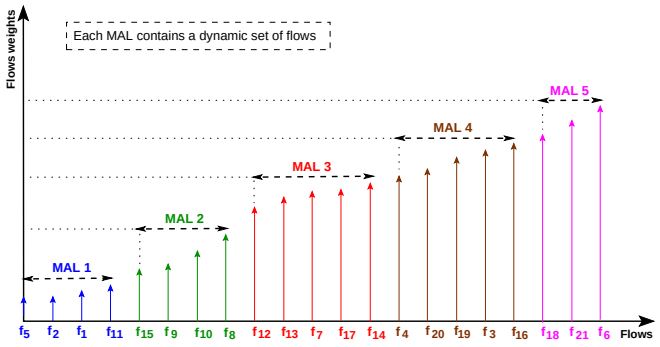


Figure 4: An illustration of the Multicast Allocation Layer (MAL) scheme

Given that we do queuing using per-packet manner and not per-bit manner, MFQ tries to guarantee that the maximum number of packets allocated to each active fbw i remains always less than the integer value of its expected allocation **in term of the capacity fraction** w_i multiplied by $qlim$, the maximum queue size in packets, i.e.; $\lfloor w_i * qlim \rfloor$. However, two fbws that have different weights may have the same maximum number of packets allowed to be queued. In the example given in Figure 4, the maximum number of allowed packets of fbw number 3 and fbw number 20 is equal to 4 because $\lfloor w_3 * qlim \rfloor = \lfloor w_{20} * qlim \rfloor = 4$ where $w_3 = 0.07$, $w_{20} = 0.065$ and $qlim = 64$. To guarantee a more fine-grained queuing, we enhance the buffer management module by using the MAL scheme that we have described above.

For every arriving packet, the router starts by identifying the multicast fbw and the MAL to which it belongs. Let fbw number i be the fbw to which belong the arriving packet and j be the number of its MAL. We generate a random value $u \in [0, 1]$ and we allow the fbw i gets **one more packet** than its MAL value if $u < w_i / MAL[j].maxAllocation$, where $MAL[j].maxAllocation$ is the maximum weight of fbws belonging to the MAL number j. As consequence, we ensure that each two fbws that belong to the same MAL will get **randomly and proportionally** to their fair share one more packet than their MAL values and we ensure a much more fine-grained bandwidth sharing. If the packet was accepted, we update its MAL state.

For completeness, we give the pseudo-code of the algorithm in Appendix A.

5.3 MFQ and Layered Multicast

When evaluating new network control mechanisms one must evaluate the impact of the proposed mechanisms on application performance. We discuss in this section the MFQ impact on layered multicast application performance.

When the multicast session using a layered transmission scheme, the data is split into layers and each layer sends to a different group address. Depending on the loss rate seen by the receivers, they join and leave layers to adapt to the network situations. We demonstrate how MFQ can achieve a priority dropping without explicitly assigning priorities to the transmission layers.

Assuming a multicast source decodes data into n transmission layers and that there are R_i receivers subscribed to the layer l_i (l_0 is the base layer). Given that receivers who join the layer l_i should join all lower layers $l_0 ... l_{i-1}$, we can easily write the following inequality: $R_{n-1} \leq ... \leq R_j \leq R_{j-1} \leq ... \leq R_1 \leq R_0$.

Without loss of generality, we assume the use of the LogRD function to allocate the bandwidth fairly between multicast fbws. The weight of the transmission layer number j is $w_j = \frac{1+\ln R_j}{\sum_{p=0}^{p=n}(1+\ln R_p)}$. We can easily write the following inequality:

$$w_{n-1} \leq ... \leq w_j \leq w_{j-1} \leq ... \leq w_1 \leq w_0. \quad (8)$$

We can then write:

$$F_{MAL}(f_{n-1}) \leq ... \leq F_{MAL}(f_j) \leq F_{MAL}(f_{j-1}) \leq ... \leq F_{MAL}(f_1) \leq F_{MAL}(f_0), \quad (9)$$

where $F_{MAL}(f_i)$ is the MAL to which belongs the flow associated to the transmission layer number i. Thus, it is clear that layers with lower number of receivers (lower priority layers) will see a loss rate higher than those with higher number of receivers and in particular the base layer l_0 (highest priority layer).

Similar approaches that need a network-support including priority dropping [1] schemes require that the network support as many loss priority levels as layers. In addition, to ensure a fair allocation of network resource, they also require that each session uses the same set of priorities than others. Furthermore, priority dropping provides no incentives for receivers to lower their subscription level.

The development of a multicast-congestion control mechanism that uses MFQ to ensure a priority dropping between flows corresponding to the transmission layers is one of our future works that we will investigate.

6 Complexity and Implementation Issues

The deployment of 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:

- 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
- 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 a large number of routers and many research works such as [25] demonstrate its deployment feasibility.

At each "SBQ router", we need to maintain a state per active multicast flow. Upon a packet arrival, the router needs to (1) determine the flow and the MAL number to which the arriving packet belongs, (2) update the flow state and the MAL state parameters such as the number of packets of the corresponding flow and the allocation of this flow provided by the multicast bandwidth sharing module. As shown in [25], these operations and even the packet classification could be efficiently implemented because it only consists of reading the flow ID for IPv6 or the pair (source IP address, multicast destination address) for IPv4.

At branch points in the multicast tree, SBQ routers must record additional information needed by the multicast bandwidth allocation module such as the number of downstream receivers. The processing complexity at SBQ routers is increased in two (minor) ways. First, during the lookup-operation for each packet arrival, useful information must also be retrieved from the routing table entry. Second, before accepting the datagram, SBQ should verify that the flow is authorized to get more packets in the queue. While our mechanism would benefit from better bandwidth allocation functions, it is explicitly designed to be robust to coarse implementation of the inter-multicast fairness function using the MAL scheme described in Section 5.2.2.

Additionally, SBQ can also interact with window-based as well as rate-based multicast congestion control by sending back to the source an ECN-like (ECN - Explicit Congestion Notification) or another specific message in order to signal its fair rate. This issue is out of the scope of this paper.

It should also be noted that the source address and the destination group address are not only needed by the MFQ mechanism but also by the multicast routing lookup module which has to determine the list of outgoing interface(s) for each incoming multicast packet.

One major advantage of our approach is that it minimizes the complexity of designing multicast congestion control schemes. 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.

7 Simulation Methodology and Results

We have implemented the SBQ scheduler in the ns-2 network simulator [23] and we conducted several experiments to evaluate its performance for different traffic characteristics.

7.1 Single Bottleneck Link

We validate our scheme for a topology consisting 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. As shown in Figure 5, sources are connected to router n_1 and all destinations are downstream to router n_2.

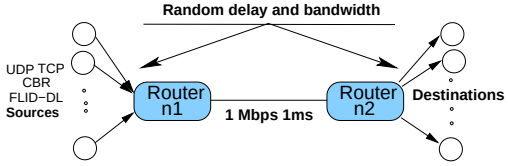


Figure 5: A single congested link simulation topology. The congested link has a capacity of 1 Mbps and 1ms propagation delay.

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 consider responsive and non-responsive sources and heterogeneous links with different delay and bandwidth.

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) [5] sources. As we have outlined earlier, the protocol FLID-DL uses a Digital Fountain [6] at the source in which the sender encodes the original data and redundancy information such that receivers can decode the original data once they have received a fixed number of arbitrary but distinct packets. The FLID-DL simulation parameters are the same as those recommended in [5][9].

Each source uses 17 layers encoding, each layer is modeled by a UDP traffic source. In Tab. 1, we give the values of different parameters used.

As outlined earlier, we use MFQ 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 [18] 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.

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

Table 1: FLID-DL parameters used in simulation

Parameter	Description	Value
c_mult_	the multiplicative factor	1.3
slot_time_	the slot time	0.5
number_of_layers_	the number of layers	17
simulated_rtt_	for tcp rate value calculation	0.1
rng_speed_	the random generator's speed	0
packet_payload_	the number of bytes in packet	1000

The 32 unicast sources are composed as follows:

- Flows from 33 to 48: UDP sources. These unicast sources transmit packets at different constant bit rates (CBR unicast sources).
- Flows from 49 to 64: TCP sources. Our TCP connections use the standard TCP Reno implementation provided with ns-2 network simulator.

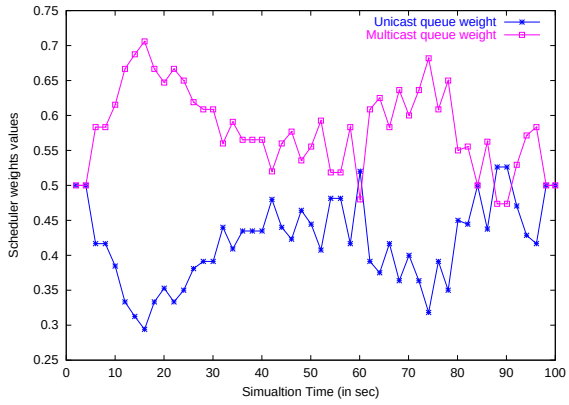


Figure 6: Scheduler weights variation in function of the simulation time

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

- 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 5 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 CBR 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. Note that the active queue management mechanism used in the unicast queue is the responsible for protecting TCP connections against bursty unicast sources. MFQ is able to protect multicast responsive flows (FLID-DL flows) because, as we have explained above, it uses a pre-defined threshold and it privileges accepting packets belonging to new flows. Initially (at $t = 0$ sec), the weight X of the SBQ scheduler (see Eq. 4) is set to 0.5.

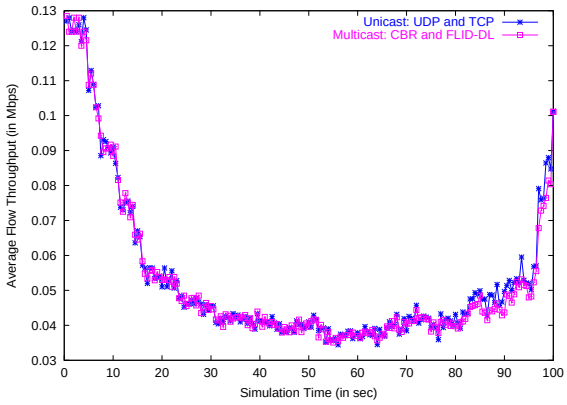


Figure 7: The variation of the average unicast and multicast average rate in function of the simulation time over 500 msec

In Figure 6, 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 two queues of the SBQ scheduler. The multicast weight increases when the unicast weight decreases and vice versa.

We look at 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 and multicast rates over $500\ msec$ of simulations. We show in Figure 7, the variation of the unicast and the multicast average rate in function of the simulation time.

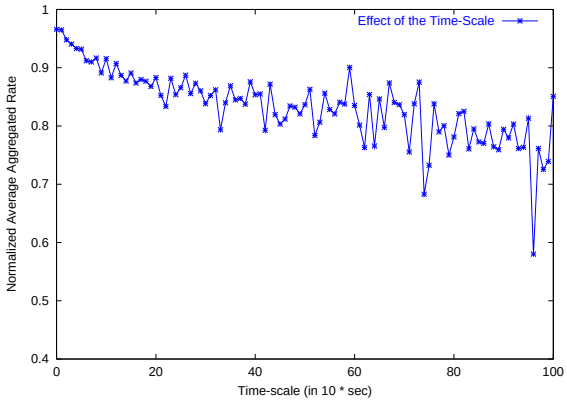


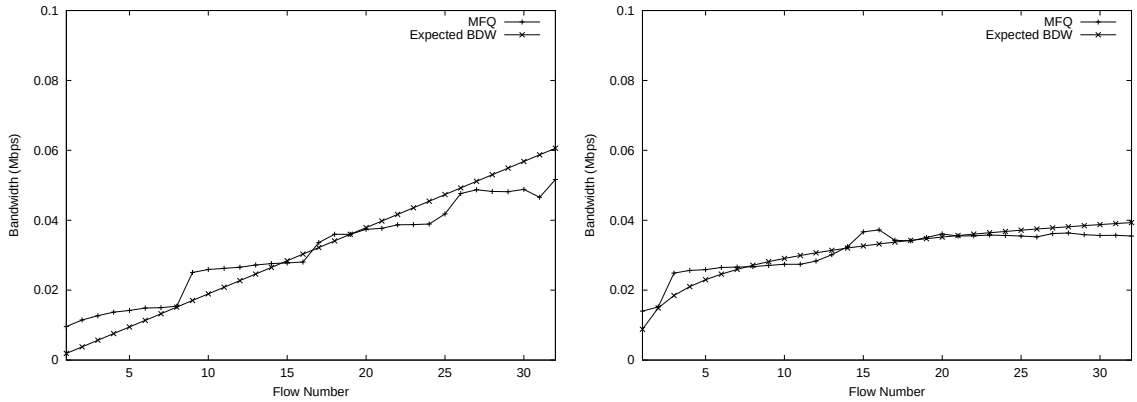
Figure 8: Sensitivity of SBQ performance on the time-scale value

As we can observe, the results match exactly what we expect. Indeed, two main observations can be derived from the plots of Figure 7. Firstly, there is a fluctuation on the average aggregated rate for unicast and multicast flows which due to the random start and finish time of all 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 8, we show the variation of this metric in function of the time-scale value. We can conclude that using a 1 sec time-scale allows to 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.

In a new experiment, we use the network configuration of Figure 5 and we suppose that the flow i has exactly i receivers. We set the start time of all flows to 0 and their finish time to 100 seconds. We plot in Figure 9(a) and Figure 9(b) the obtained and the expected multicast bandwidth sharing for linear and logarithm bandwidth allocation policy, respectively. As show in these figures, MFQ matches closely the fair share for both policies despite the heterogeneity of the multicast sources.



(a) Using a linear allocation function

(b) Using a logarithm allocation function

Figure 9: 32 multicast flows. Flows form 1 to 16 are FLID-DL sources and those from 17 to 32 are CBR sources

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 claim, we compare in Figure 10 the normalized rate over $500\ msec$ of simulation time for linear (LIN), logarithm (LOG), and receiver-independent (RI) bandwidth sharing policies which are defined in [18][10]. As we can see the normalized average rate is always varying around 1 (between 0.82 and 1.18) for the three cases.

We study the bandwidth shared among multicast flows in the multicast queue of the SBQ scheduler provided by the 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 and we vary the number of UDP unicast flows from 1 to 16 flows. In Figure 11, we show the bandwidth fair rate obtained for the 32 multicast flows. It is very clear from the plots that the shape of flows rate curves follows a logarithm function of the number of downstream receivers given that it was assumed to be equal to the flow index.

7.2 Multiple Bottleneck Links

In this sub-section, we analyze how the throughput of multicast and unicast flows is influenced when the flow traverses L congested link. We performed two experiments based on the topology of Figure 12. We index the links from 1 to k.

[10] Assume 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.

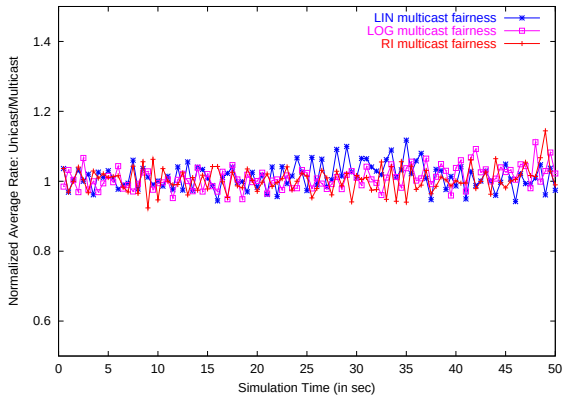


Figure 10: The normalized rate when modifying the intra-multicast fairness function over 500 msec of simulation time

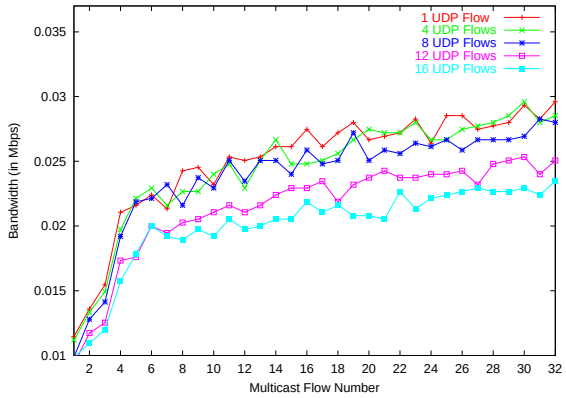


Figure 11: Rates of multicast fbws when using a logarithm multicast bandwidth sharing function

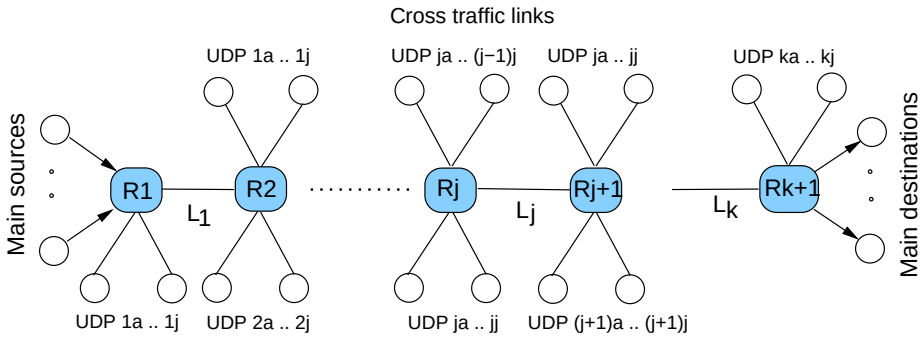


Figure 12: Topology for analyzing the effects of multiple congested links on the performance of SBQ

We use exactly the same traffic parameters as the case of single bottleneck described above and we measure the aggregated bandwidth received by each service type (unicast or multicast) in function of the number of congested links. Again, a link L_j, $2 \leq j \leq 10$, is kept congested by setting its capacity C_j to $C_{j-1} - 50$ Kbps. The capacity C_1 of the link number 1 is set to 1 Mbps. 10 cross CBR sources send at 200 Kbps on each of the congested links. This cross traffic enter the path in one of the router and exists at the next.

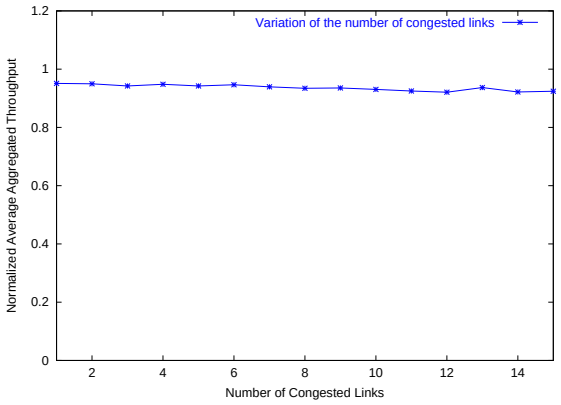


Figure 13: The normalized rate as a function of the number of congested links

We plot in Figure 13, 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 increases.

We also extended the unicast fairness index introduced by Jain in [17] to multicast traffic. Instead of the expected unicast fair rate, which is always the same for unicast connections, we use that of multicast fbws which may differ from one fbw to another depending on the fairness function implemented in the bandwidth allocation module. The **multicast fairness index** is then computed as follows:

$$1 - \frac{1}{n} \sum_{i=1}^{i=n} \left| \frac{t_i - \overline{t_i}}{\overline{t_i}} \right| \tag{10}$$

where n is the number of active multicast fbws, $\overline{t_i}$ and t_i are the expected and the obtained bandwidth share of the multicast fbw i, respectively. We plot in Figure 14, the variation of the multicast fairness index as a function of the number of congested links. As we can see, the index value remains close to 1 even when the number of congested links increases for RI, LIN, and LOG fairness functions. As expected in [18], the LOG fairness function has better fairness index variation than the two other functions.

8 Conclusions and Future Works

In this paper, we have presented a CBQ-like scheduler for bandwidth sharing between unicast and multicast fbws. 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 detailed in this paper and which provides the expected multicast bandwidth sharing between multicast fbws 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 fbws according to our new definition of fairness.

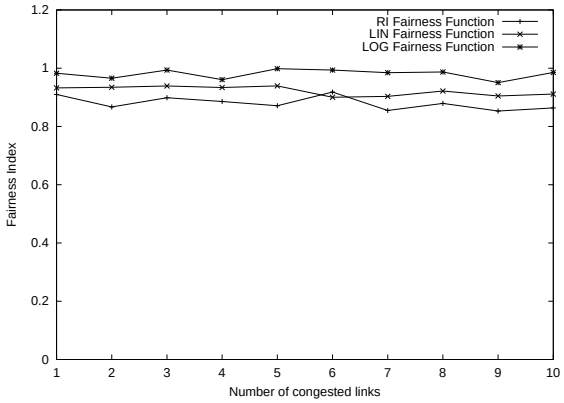


Figure 14: MFQ performance in multiple bottleneck link

Future work could evaluate the performance for other types of multicast traffic that include others application-based or transport-based congestion control mechanisms. Indeed, there is a big debate going on in today's the network research community on whether "end-to-end"argument is still the golden rule, or just dead [7]. The current Internet is built on top of the "end-to-end" philosophy. End systems are responsible for constructing better than best-effort service, therefore the network is often drawn as a big cloud with end system (PCs, etc) attached to it, with its inner workings hidden from the end users. However, this is not always true in today's Internet, where servers and proxies for specific applications (the web, mainly) are installed inside the network in attempt to improve the application performance. Therefore many people believe the "end-to-end" service model is dead, since lots of services can be done better inside the network than by the end system.

In a multicast network, it is very important that the bandwidth is allocated fairly among the multicast sessions. An appealing approach that accommodates heterogeneous receivers and adapts to congestion is to encode the data onto multiple layers and transmit each layer on its own multicast group.

Layered multicast protocols typically use a receiver-driven approach in which the end-systems decides which layers should be delivered [5, 27].

An alternative approach is to use a SBQ scheme where the network, rather than the receiver, decides which layers (fbws) should be delivered. When congestion arises, the network drop the least important packet (e.g; that has the less number of downstream receivers) first. An important benefit of SBQ is stable and fair allocation of bandwidth. Because packet losses are concentrated at the highest layer(s) (given their lower number of receivers), the highest layers absorb the majority of transient losses caused by short term congestion.

We do not claim that such type of mechanism can be easily deployed in the today's Internet, however minimizing the complexity of developing multicast congestion control mechanisms between the end hosts and the network can encourage the development of added-value multicast applications. Indeed, SBQ allows the ISPs to differentiate between unicast and multicast fbws according to a pre-defined multicast bandwidth allocation function and a novel unicast and multicast fairness definition (inter-service fairness) which guarantees that each multicast fbw remains globally tcp-friendly.

Acknowledgments

The author would like to thank the anonymous reviewers for valuable, detailed, and careful comments and feedback.

References

[1] S. Bajaj, L. Breslau, and S. Shenker, *Uniform versus priority dropping for layered video*, In Proc. of SIGCOMM'98, pp. 131-143, September 1998

[2] T. Bates, et al., *Multiprotocol Extensions for BGP-4*, IETF, RFC 2283, February 1998.

[3] D. Bertsekas and R. Gallagher, *Data Networks*, Englewood Cliffs, NJ, Prentice-Hall, 1992.

[4] S. Blake, et al., *An Architecture for Differentiated Services*, IETF, RFC 2475, December 1998.

[5] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver, *FLID-DL: Congestion Control for Layered Multicast*, In Proc. of NGC 2000, pp. 71-81, November 2000.

[6] 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.

[7] D. Clark and B. Marjory, *Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World*, ACM Transactions on Internet Technology (TOIT), Volume 1, Issue 1, pp. 70-109, August 2001.

[8] A. Clerget, and W. Dabbous, *TUF : Tag-based Unified Fairness*, In Proc. of IEEE INFOCOM 2001, April 2001.

[9] S. Deering, *Multicast routing in a datagram internetwork*, Ph.D. thesis, 1991.

[10] S. Deering, *Host Extensions for IP Multicasting*, IETF, RFC 1112, August 1989.

[11] F. Filali, H. Asaeda, and W. Dabbous, *Counting the Number of Members in Multicast Communication*, in Proc. of the Fourth International Workshop on Networked Group Communication (NGC 2002), Boston, USA, October 2002.

[12] F. Filali, L. Fazio and W. Dabbous, *Enabling Unicast and Multicast Differentiation in DIffServ Architecture*, Submitted for publication.

[12] 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.

[13] S. Floyd and V. Jacobson, *Link-sharing and Resource Management Models for Packet Networks*, IEEE/ACM TON, V. 3, No.4, 1995.

[14] S. Floyd, M. Handley, J. Padye, and J. Widmer, *Equation-based congestion control for unicast applications*, In Proc. ACM SIGCOMM'00, August 2000.

[15] T. N.H. Henderson and S. N. Bhatti, *Protocol-independent multicast pricing*, In Proc of NOSS-DAV'00, June 2000.

[16] H. Holbrook and B. Cain, *Source-Specific Multicast for IP*, IETF, Internet draft, draft-ietf-ssm-arch-01.txt, November 2002.

[17] R. Jain, *The art of computer systems performance analysis*, John Wiley and sons QA76.9.E94J32, 1991.

[18] 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.

[19] D. Lin and R. Morris, *Dynamics of Random Early Detection*, In Proc. of ACM SIGCOMM'97, September 1997.

[20] M. Luby, V. Goyal, and S. Skaria, *Wave and Equation Based Rate building block*, IETF, Internet draft, draft-ietf-rmt-bb-webrc-04.txt, December 2002.

[21] M. Luby, L. Vicisano, and A. Haken, *Reliable Multicast Transport Building Block: Layered Congestion Control*, IETF, Internet draft, draft-ietf-rmt-bb-lcc-00.txt, November 2000.

[22] A. Mankin, et al., *IETF Criteria for evaluating Reliable Multicast Transport and Applications Protocols*, IETF RFC 2357, June 1998.

[23] S. McCanne and S. Floyd, *Ucb/lbnl/vint network simulator (ns) version 2.1b6*, `http://www-mash.cs.berkeley.edu/ns/`, June 2000.

[24] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, *Modeling TCP throughput: a simple model and its empirical validation*, In Proc. of ACM SIGCOMM, Vancouver, Canada, September 1998.

[25] 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.

[26] 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.

[27] L. Vicisano, L. Rizzo, and J. Crowcroft, *TCP-like congestion control for layered multicast datda transfer*, In Proc. of IEEE INFOCOM'98, San Francisco, CA, March 1998.

[28] J. Widmer and M. Handley, *TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification*, IETF, Internet draft, draft-ietf-rmt-bb-tfmcc-00.txt, November 2001.

Appendix A: The pseudo-code of MFQ mechanism

Variables:

```
fs_ :  Flow State {receivers, active, qlen, fairness}
ls_ :  Layer State {maxFlowID, maxAllocation }
```

Algorithm:

```
/* determine the multicast flow ID */
flowID ← classify(pkt)

/* update the flow state if needed*/
if (flowID is not active) {
  fs_[flowID].receivers ← getReceivers(p)
  fs_[flowID].count++
  newflow ← 1
} else (if the number of receivers has changed)
  update the number of receivers and the total number

/* get the expected allocation */
weight ← getAllocation(flowID])

/* determine the MAL to which belongs the current flow */
int mal_ ← max(1,(int)(weight*thrsh))

/* compute the number of packets to add to the expected allocation {0 or 1} */
double u ← Random::uniform();
if (u > f / ls_[mal_].maxAllocation )
  p ← 0;
else p ← 1;

/* update the mean queue size */
```

```
update the mean queue size(qlen)

/* the MFQ dropping decision as explained in Section 5.2 */
if (qlen >= qlim)
  if (!  newflow ) drop(pkt)
  else{
    pktd ← getPacketToDrop()
    update_mal_state(pktd)
    update_flow_state(pktd)
    drop(pktd)
    enqeue(pkt)
  }
else {
if (fs[flowID].qlen < mal_ + p)
  if ( !  newflow && qlen > thrsh ) drop(pkt)
  else enqeue(pkt)
else
  drop(pkt)
}

/* update the flow and the MAL states if the pkt was not dropped */
if (pkt was not dropped) {
  update_mal_state(pkt)
  update_flow_state(pkt)
}
```

Appendix B : The algorithm for counting unicast connections

Connection count(packet p)
h = H(p)
if $v(h) = 0$
$v(h) = 1$
$N = N + 1$
$t =$current time
n_{clear} = $v_{max} \frac{t - t_{last}}{t_{clear}}$
if n_{clear} > 0
t_{last} =t
for $i = 1$ to n_{clear} - 1
r = random(0...v_{max} - 1)
if $v(r) = 1$
$v(r) = 0$
$N = N + 1$
return (N)

Variables:

- $v(i)$ Vector of v_{max} bits. $v(i)$ indicates if a packet from a connection with hash i has arrived in the last t_{clear} seconds.
- N Count of one bits in v.
- t_{last} Time at which bits in v were last cleared.
- r Randomly selected index of a bit to clear in v. Constants: v_{max} Size of v in bits; should be larger than the number of expected connections.
- t_{clear} Interval in seconds over which to clear all of v.
- $H(p)$ Hashes a packets connection identifying fields to a value between 0 and v_{max}.