A New Bandwidth Sharing Scheme for Non-Responsive Multicast Flows

Fethi Filali and Walid Dabbous
INRIA, 2004 Route des Lucioles, BP-93,
06902 Sophia-Antipolis Cedex, France
Telephone: +33 4 92 38 78 25
Fax: +33 4 92 38 79 78

***Abstract*— In this work, we propose a new active queue management mechanism called MFQ (Multicast Fair Queuing) for multicast flows. MFQ is independent of the inter-multicast fairness policy and it interacts with a multicast bandwidth allocation module that can implement either a multicast fairness policy or a multicast pricing model and tries using a single FIFO queue to achieve the expected allocation which may depend on the number of downstream receivers.**

We perform simulations to evaluate the performance of MFQ for different bandwidth allocation schemes. Results obtained suggest that MFQ achieves the expected allocation for non-responsive multicast flows.

***Index Terms*— Bandwidth Sharing, Inter-Multicast Fairness, Non-Responsive Multicast Flows, Active Queue Management**

I. INTRODUCTION

MANY recent research work believe that the definition of the inter-multicast fairness (fairness between multicast flows) should take into account the number of competing groups, the number of flows per group, and the number of receivers per flow[1]. In particular, Legout et al. [5] have defined three different allocation strategies as well as criteria to compare these strategies. They showed that the LogRD policy[2] always leads to the best tradeoff between receiver satisfaction and fairness. To implement this proposition in real networks, they proposed to introduce their allocation scheme to the general scheduler by configuring the weights of a GPS scheduler [8], to achieve the goal. This goal can also be met by reserving the bandwidth in the network for either individual connections or group of connections, and explicitly allocating network bandwidth on a packet-by-packet basis by scheduling packets across network links. However, the two methods are complex and require either the use of Fair Queuing mechanisms [1] [8] in each router or the use of RSVP-like bandwidth reservation signaling protocols which needs a close coordination and integration between all routers (and hence all network providers) along the path from sender to receiver.

[1]Throughout this paper, we mean by a receiver the designed router of one or more multicast members in its directly attached LAN(s).

[2]The LogRD policy consists to give to the multicast flow 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.

In this paper, we are investigating an alternate approach based on active queue management rather than packet scheduling or explicit bandwidth reservation. We are developing a new active queue management mechanism for routers that provides the expected bandwidth allocation between non-responsive multicast flows.

We call our Active Queue Management (AQM) mechanism, MFQ, for Multicast Fair Queuing, a per-flow dropping mechanism which interacts with a bandwidth allocation module that provides for each multicast flow its expected bandwidth. MFQ belongs to the class of per-flow dropping like FRED (Flow Random Early Drop) [6]. The operations done by MFQ are not as complex as manipulation of priority queues like FQ (Fair Queuing) [1], given that they only consist of dropping or queuing the packet. In addition, it was shown in [2], [9] that the class of per-flow dropping algorithms and even FQ can be cost effectively implemented in high-speed backbone routers. So, one of the initial assumptions of this paper is that it is possible to use FRED-like architecture to provide cost-effective service guarantees at sufficiently high speeds.

As we know there is no similar work in this area. We are developing an innovative solution to guarantee the inter- and intra- multicast fairness in the network. In absence of similar approaches, we compare MFQ performance with analytical-computed expected results.

We use simulations to evaluate the effectiveness and performance of MFQ for different network configuration scenarios. In this paper, we present only the performance of our mechanism for non-responsive multicast sources; i.e., sources that don't modify their behavior when a packet loss occurs. We show that the bandwidth share obtained using MFQ is very close to the expected allocation for both linear and logarithm bandwidth allocation scheme.

The remainder of this paper is organized as follows. Section 2 details MFQ, our per-flow dropping mechanism, and its main components. We show the simulation results for different sources configuration and bandwidth allocation strategies in Section 3. Section 4 concludes this paper by summarizing results and outlining future work.

0-7803-7400-2/02/$17.00 © 2002 IEEE

II. MFQ: MULTICAST FAIR QUEUING

A. MFQ Architecture

Two modules compose our MFQ mechanism:

- The multicast bandwidth allocation module which provides as output the expected share for each active multicast flow. It may implement either a multicast fairness function [5] or a multicast pricing model [4].
- The buffer management module which uses a single FIFO queue and interacts with the first module to decide the drop preference.

We explore hereafter separately these two modules.

1) Multicast Bandwidth Allocation Module: The Internet today is an inter-connection of Autonomous Systems (AS) (also called domains). We assume that network operators belonging to each AS have a **single** and **clearly defined** multicast bandwidth sharing policy configured by its administrative authority. This policy can be configured some routers inside the AS instead of explicit bandwidth reservation. The multicast bandwidth allocation module returns the link capacity fraction allocated for the requested flow.

We should point out that the receivers belonging to different ASs using different multicast bandwidth allocation policies may receive data with different quality of service even if there is no more competing flows in the delivery tree.

We develop hereafter a first framework of the multicast bandwidth allocation module. As we have pointed our 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. Using the network model introduced in Section **??**, we define the bandwidth λ_{ij}, in bits per second, allocated to group G_i, in link l_j as following:

$$\lambda_{ij} = F_1(G_i) * C_j \quad (1)$$

where $F_1(.)$ is the ***inter-group multicast fairness function*** that defines the way how the bandwidth should be shared among active multicast groups[3]. In other words, $F_1(G_i) * C_j$ is the maximum bandwidth share has to be allocated to group G_i in link l_j. $F_1(.)$ is the bandwidth share function at the group level.

The bandwidth allocated to flow k of the multicast session $M_k \in G_i$, in link l_j is given by the following expression:

$$\lambda_{kj} = F_2(M_p/M_p \in G_i) * \lambda_{ij} \quad (2)$$

where λ_{ij} is computed using Eq. 1. The function $F_2(.)$ returns the fraction of the bandwidth which has already been allocated to group G_i and that should be given to the session M_k. We call this function the ***intra-group multicast fairness function***. This function depends on the number of the downstream receivers in each multicast session M_p that belongs to the same group G_i. $F_2(.)$ is the bandwidth share function at the flow (source) level.

We should note also that the functions $F_1(.)$ and $F_2(.)$ should satisfy the following properties:

- for each multicast group G_i: $0 < F_1(G_i) < 1$, and $0 < F_2(M_p) < 1$ for each $M_p \in G_i$.
- in each link l_j: $\sum_i F_1(G_i) = 1$, and $\sum_{pM_p \in G_i} F_2(M_p) = 1$ 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 our mechanism can adapt itself according to the bandwidth allocation function used, the number of receivers per flow, the number of active flows, and the number of active multicast groups. In addition, the bandwidth sharing remains very close to the expected allocation allowed by the multicast bandwidth allocation module.

2) Buffer Management Module: The role of the buffer management module is to make the queuing/dropping decision of the incoming multicast packet. In MFQ design phase, we have conducted to take many decisions to be independent of the network and sources characteristics and especially the following parameters:

- the change on the number of downstream receivers,
- the source behavior when a packet is lost,
- the sources rates.

We use a single FIFO queue with a pre-configured maximum size in packets. A multicast flow is considered instantaneously inactive if there is no packet in the queue belonging to this flow. The multicast flow state contains:

- the number of packets belonging to this flow and which are waiting in the queue,
- and the current expected bandwidth allocation.

Considering 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 flow i remains always less than the integer value of its expected allocation **in terms of the capacity fraction** f_i multiplied by $qlim$, the maximum queue size in packets, i.e.; $(integer) f_i * qlim$. However, two flows that have different expected allocation may have the same maximum number of packets allowed to be queued. For example, suppose that $f_{10} = 0.019048$, $f_{15} = 0.028571$ and $qlim = 64$, we obtain $(integer)0.019048 * 64 = (integer)0.028571 * 64 = 1$. Hence, the flows number 10 and number 15 will have the same maximum number of packets. To guarantee a more fine-grained queuing, i.e., an allocation being as close as possible to the expected allocation, we introduce the notion of a Multicast Allocation Layer (MAL).

A MAL is a set of flows that have the same or different expected allocation in term of the link capacity fraction but the same allocation in term of maximum number of packets allowed to be queued. In each MAL state, we maintain the multicast flow ID which has the maximum allocation and the value of this allocation. Based in this definition, flows 10 and 15, given in the previous example, belong to the same MAL number 1.

For every arriving packet, the router starts by identifying the multicast flow and its MAL and updating the flow state and its MAL state only if the flow is a new active flow. If the flow is

[3] A multicast group may have one or more sessions (flows) from different sources that share the same communication link.

already active and there is a change on the number of receivers, we ask the bandwidth allocation module for the new value of the expected allocation of this flow.

Let's flow number i be the flow to which belong the arriving packet. We generate a random value u in $[0, 1]$ and we allow that each flow gets **one more packet** than its allocation layer if $u < f_i/MAL[j].maxAllocation$, where $MAL[j].maxAllocation$ is the maximum allocation value (in term of link capacity fraction) of flows belonging to the MAL number j. As result, we ensure that each two flows that belong to the same multicast allocation layer will get **randomly and proportionally** to their expected allocation one more packet than their multicast allocation layer and so that a much more fine-grained bandwidth sharing.

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

We use the same method as RED (Random Early Drop) [3] to estimate the queue mean 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[4].

If the mean queue size is between the maximum queue size and the threshold *thrsh*, we accept the packet only if it belongs to an inactive flow. If the queue size is equal or greater that the maximum queue size, we drop the incoming packet if its flow is already active (there is at least one packet belongs to this flow in the queue), otherwise we drop randomly a packet from the queue and we queue the incoming packet. By this way, we allow to the new flow to be active and we remove the bias against bursty sources.

III. Simulation Methodology and Results

We have examined the behavior of MFQ under a variety of conditions. We use an assortment of traffic sources and topologies. Due to space limitations, we only report on a small sampling of the simulations we have run[5]. All simulations were performed in ns-2 [7], which provides accurate packet-level implementation for various network protocols. All algorithms used in the simulation, expect MFQ, were part of the standard ns-2 distribution[6].

Without loss of generality, we evaluate MFQ mainly for three different multicast bandwidth allocation schemes which are proposed in [5]: receiver independent, linear, and logarithm allocation function.

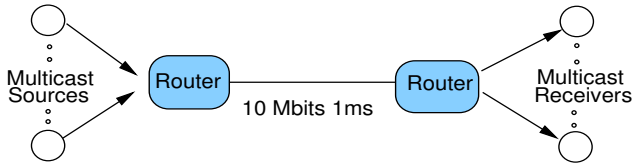

Fig. 1. A single congested link simulation topology. The congested link has a capacity of 10 Mbps and 1ms propagation delay.

We start by validating MFQ for a simple topology consisting of a single congested link connecting two routers $n1$ and $n2$ and having a capacity C equal to 10 Mbits/sec and a propagation delay D equal to 1 ms. As shown in Figure 1, the multicast sources are connected to router $n1$ and the router $n2$ belong to the receivers to.

We use different bandwidth allocation functions to evaluate MFQ. We compare the expected bandwidth with that obtained by MFQ for 32 multicast flows. The maximum buffer size $qlim$ (the same meaning as the maximum queue size) used in all our simulations is set to 64 packets.

We index the multicast flows from 1 to 32 and we compute the bandwidth share depending on the allocation function used. When we use a linear allocation function, the fair share of flow i is equal to $\frac{i}{\sum_{j=1}^{j=32} j} * C$ and it is equal to $\frac{1+\lg i}{\sum_{j=1}^{j=32}(1+\lg j)} * C$ in the case of logarithm allocation function.

In this section, we assume that each multicast source is a non-responsive CBR source; i.e.; without any type of congestion control mechanism (neither application-level nor transport-level). For this case, we use a CBR generator simulating a real-time audio application. We assume that the source i sends data at a rate equal to i more that its expected fairness rate provided by the fairness function. For example, there is a total amount of $\sum_{i=1}^{i=32} \frac{i}{32} * C = 16.4 * C$ kbps arriving at the bottleneck link when we use a receiver-independent multicast fairness function. Thus, the flow 1 sends 0.3125 *Mb/sec*, and flow 2 sends 0.625 *Mb/sec*, and so on.

Unless otherwise specified, each simulation lasts 30 seconds, the source number i starts sending data $i * 0.001$ sec after the simulation starting time and the packet size is assumed to be equal to 1000 bytes. In addition, the flow number i has i downstream receivers.

We are first interested in the convergence phase of MFQ. We use a linear bandwidth allocation function and we plot in Figure 2, the variation of the bandwidth share of some selected flows in function of the simulation time. We can see that MFQ reaches a stable sate after one second only of simulation for all the flows. In addition, the flow number i gets more bandwidth share than flows $1..(i-1)$ which exactly what we expect giving that we don't use a receiver-independent fairness function.

In all the following simulations, we start the measurements one second after the simulation starting time so that, the actual queue size and the average queue size have already reached a

[4]The value of W_q is set to 0.002 in all simulations.

[5]A fuller set of tests, and the scripts used to run them, are available at `http://www.inria.fr/rodeo/filali/mfq`.

[6]The MFQ mechanism and the generators used in our simulation as well as the script can all be downloaded from `http://www.inria.fr/rodeo/filali/mfq`. The Linux implementation of MFQ will also be available shortly.

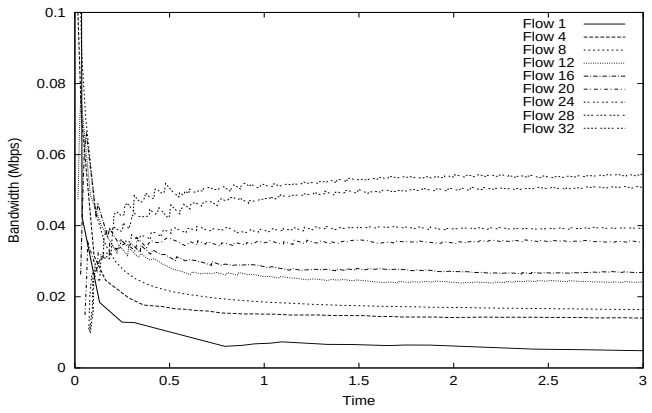

Fig. 2. Convergence of the MFQ for some multicast flows

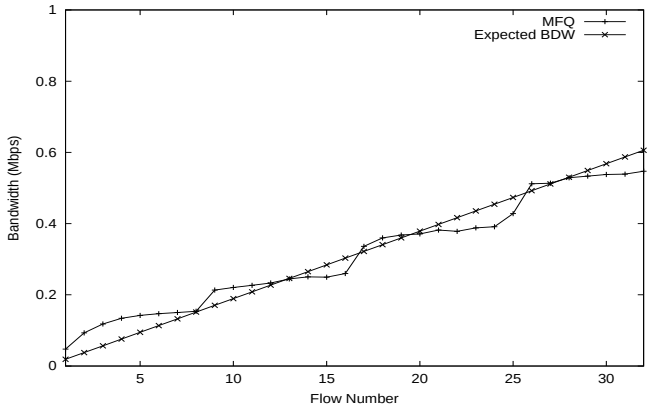

Fig. 4. 32 CBR multicast flows using a linear multicast allocation function

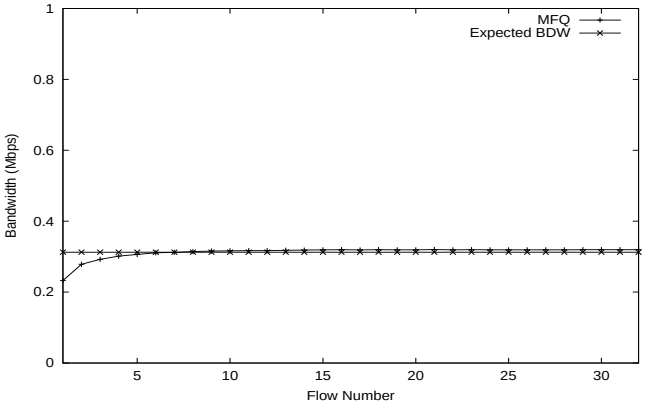

Fig. 3. The bandwidth share provided by MFQ when each multicast flow has exactly one receiver

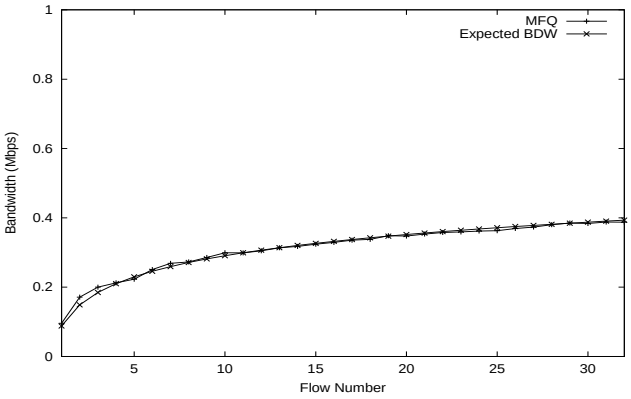

Fig. 5. 32 CBR multicast flows using a logarithm multicast allocation function

stable state.

Figure 3 shows the bandwidth share where each multicast flow has only one receiver[7], emulating a unicast CBR source. Independent of the multicast bandwidth allocation function used, all the flows should get the same bandwidth share $\frac{1}{32} * C = 0.3125\ Mbits/sec$. MFQ achieves a good precise degree of fairness between multicast flows.

In the following, we assume that the flow number i has exactly i downstream receivers during the simulation. In Figure 4 and Figure 5, we plot the bandwidth share of each multicast flow when we use a linear and a logarithm allocation function, respectively. We can see that the bandwidth allocation provided by MFQ is close to the expected fairness for the two cases.

Let's now see how the use of the MAL scheme helps MFQ to reach the expected results. In Figure 6, and Figure 7, we plot the bandwidth share obtained by MFQ with and without MAL scheme. According to these results, our MAL scheme can achieve a good fine-grained bandwidth sharing. In addition, we can easily see the repartition of flows in MALs when we don't use MAL scheme. For example, in the case of linear allocation function (Figure 6), flows from 9 to 15 belong to the same MAL number 2, flows from 17 to 24 belong to the MAL number 3, and so on.

Without using the MAL scheme, flows 10 and 13 (belong to the MAL number 2) will get the same bandwidth share, which is not fairly. The use of MAL allows these flows to share the bandwidth fairly. Indeed, flow 13 gets more bandwidth share than flow 10.

In a second experiment, we assume that the number of receivers of flow number i is equal to i if $i \leq 16$ and it is equal to $i - 16$, otherwise. For all others parameters, we use the same values as the first experiment. We show in Figure 8 and Figure 9 the obtained bandwidth for each flow when we use a linear allocation function and a logarithm allocation function, respectively. These results confirm that our mechanism is independent of the number of receivers per group and it is independent of the multicast fairness policy used.

IV. Conclusion and Future Work

In this paper, we have presented MFQ a multicast active queue management mechanism that provides the expected mul-

[7]We obtain the same results when using a receiver independent bandwidth allocation function.

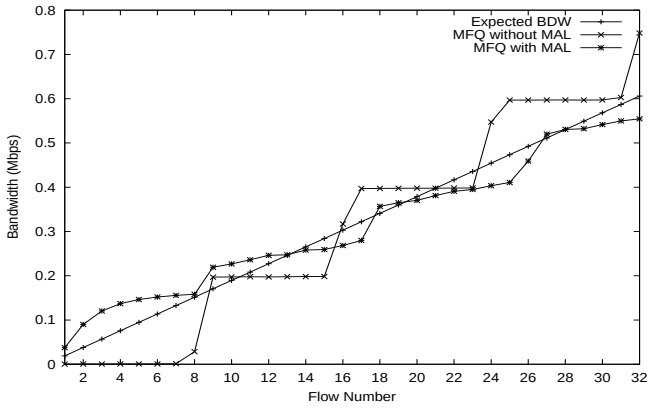

Fig. 6. 32 CBR multicast flows using a linear multicast allocation function

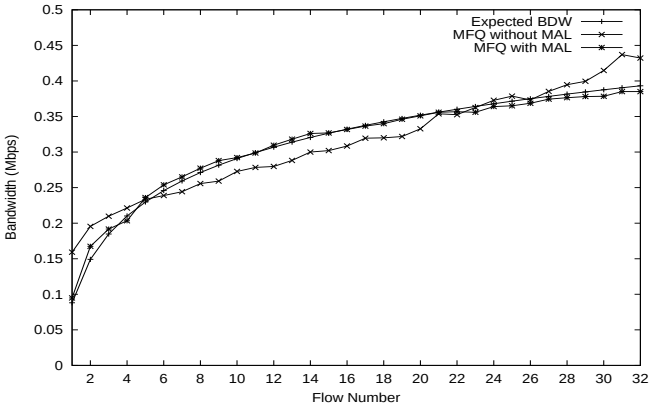

Fig. 7. 32 CBR multicast flows using a logarithm multicast allocation function

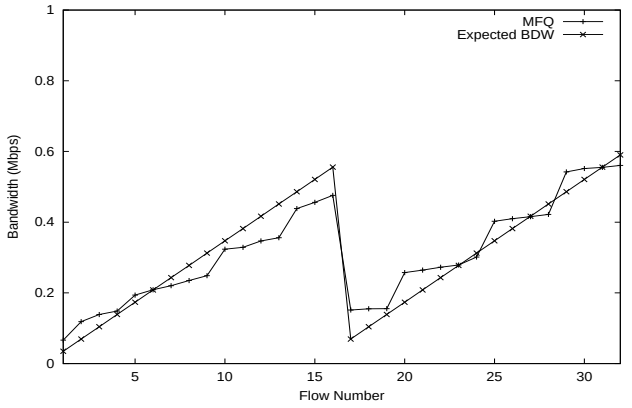

Fig. 8. 32 CBR multicast flows using a linear allocation function. The flows i and $i + 16$ have the same number of receivers which is i.

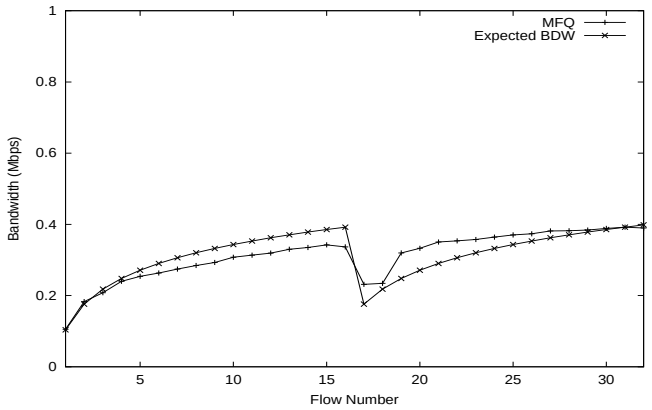

Fig. 9. 32 CBR multicast flows using a logarithm allocation function The flows i and $i + 16$ have the same number of receivers which is i.

ticast fairness policy by using a single FIFO queue. MFQ interacts with a fairness module which implements the multicast bandwidth allocation policy. The queue and the dropping decision are designed in a manner to provide a fairness close to the expected fairness. The threshold used in MFQ allows to penalize high rate multicast sources and allows a new flows to be queued.

There exist many possible areas for future work and still remain many performance aspects to be evaluated. Future work could evaluate the performance for responsive and heterogeneous multicast sources. Another area of future study is to investigate how the bandwidth should be shared between multicast and unicast flows and how MFQ should interact with the unicast queue.

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, *Analysis and Simulation of a fair queueing algorithm*, Internet working: Research and Experience, V. 1, No. 1, pp. 3-26, 1990.

[2] V. Firoiu and M. Borden, *A study of Active Queue Management for Congestion Control*, In Proc. of INFOCOM'2000, March 2000.

[3] 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.

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

[5] 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.

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

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

[8] A. Parekh and R. G. Gallager, *A generalized processor sharing approach to flow control - the single node case*, In Proc. of IEEE INFOCOM'92, v. 2, pp. 915-924, May 1992.

[9] 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.