Core Stateless Fair Bandwidth Allocation for Unicast and Multicast Flows[1]

Albert Banchs[ab], Frederic Raspall[b], David Anguera[ab] and Sebastià Sallent[a]

[a] Departament d'Enginyeria Telemàtica, Universitat Politècnica de Catalunya, Barcelona, Spain

[b] Network Laboratories, NEC Europe Ltd., Heidelberg, Germany

***Abstract*— In this paper we aim at developing a solution to fairly allocating bandwidth among unicast and multicast flows. For this purpose, we propose the *anycast max-min fairness* criterion, which is based on the well known *max-min fairness* criterion for unicast. We then present an efficient implementation of the proposed criterion: the *Multicast Fair Bandwidth Allocation* (MFBA) architecture. The MFBA architecture has two key aspects: 1) maintaining per-flow state at core routers is avoided, and 2) layered multicast is supported without the need for signaling. The proposed architecture has been implemented in a Linux PC platform. We report experimental results on the bandwidth allocation and the quality of layered video.**

***Index Terms*— Multicast, Bandwidth Allocation, Fairness, Core Stateless Queuing, MFBA, Layered Multicast, Layered Video**

I. Introduction

Multicast protocols target applications involving a large number of receivers with heterogeneous data reception capabilities. Depending on the type of application, this heterogeneity can be handled in one of two ways. In a *single-rate* session, the source adjusts its sending rate depending on the feedback it receives from the network and/or the receivers. In a typical single-rate protocol (e.g. [1]), the rate is picked up to match the bandwidth available on the most limiting data-path to a receiver.

In a *multi-rate* session, the sender can transmit at different rates to different receivers. Multi-rate sessions are typically supported through *layered multicast* [2]. With layered multicast, data is distributed among several layers, and the subset of layers that reaches each receiver can be independently determined. Layered multicast has the appealing property that the transmission rate to each receiver is constrained only by the bandwidth availability on the receiver's own data-path, and is not limited by other receivers' rate limitations in the same session. The benefits of layered multicast from a fairness viewpoint have been thoroughly studied in [3].

In this paper we develop and examine an architecture that fairly allocates bandwidth among unicast and multicast flows: the *Multicast Fair Bandwidth Allocation* (MFBA) architecture. The proposed architecture is based on previous work on core stateless fair queuing [4] in the context of unicast. With core stateless fair queuing, well-behaved flows are protected from other more aggressive or ill-behaved flows, while maintaining per-flow state at core routers is avoided.

The most notable architectural contribution of this paper is the scheme proposed to support layered multicast within the MFBA architecture. The main novelty of this approach as compared to existing work [2], [5] is that receivers are not required to signal the number of layers to be received. We believe that the absence of signaling strongly contributes to the simplicity of our solution. In addition, the scheme we propose avoids the slow response to network congestion and the instability problems of [2], [5] (see [6] for a detailed explanation of these problems).

Layered multicast is very often used for the transmission of video traffic, due to the high bandwidth requirements of this type of traffic. Our previous work in [7] showed that the performance of layered video in terms of perceptual quality does not only depend on the number of layers received but also on its evolution over time. As a consequence, it is very difficult to assess the benefits of a layered multicast scheme without analizing the resulting video signal. In order to evaluate the performance of the layered multicast support in the MFBA architecture, we have implemented the architecture in a Linux PC platform, and we have analyzed its behavior under real layered video traffic. The implementation experiences and experimental results reported are another important contribution of this paper.

The rest of the paper is structured as follows. Section II describes the *anycast max-min fairness* criterion for bandwidth allocation among unicast and multicast flows. In Section III we describe the MFBA architecture, which allocates bandwidth according to the *anycast max-min* criterion. In Section IV we propose an extension to the MFBA architecture for the support of layered multicast. Then, the Linux implementation of the MFBA architecture is described in Section V, and the experimental results obtained with the implementation are presented in Section VI. Finally, the paper closes with the conclusions in Section VII.

II. Bandwidth Allocation

A widely accepted fairness criterion for unicast flows is the well known *max-min fairness* criterion. In a max-min fair allocation, a flow cannot increase its allocated rate without causing a decrease in the rate to a more constrained flow.

Max-min fairness is achieved by equally sharing the bandwidth of a congested link among the flows contributing to congestion in the link:

$$r_i = r_j \tag{1}$$

where i and j are (unicast) flows.

The fairness criterion we have adopted in this paper for bandwidth allocation among unicast and multicast flows is the result from extending the above criterion to multicast. With our criterion, which we have called *anycast max-min fairness*, a multicast flow (*single* or *multi-rate*) and a unicast flow are entitled to the same amount of bandwidth in a congested link:

$$r_i = r_j \tag{2}$$

[1]This work has been partially funded by DFN with the ANETTE project and CICYT with the project TIC98-0495-C02.

where i and j are (multicast or unicast) flows contributing to congestion in the link.

Note that, with the above criterion, the rate to which a multicast receiver is entitled is equal to the rate that this receiver would obtain if unconstrained by the other receivers in the group, assuming max-min fairness.

Rubenstein et al. [3] had already defined the above notion of *anycast max-min fairness* for the case of *multi-rate* multicast, under the term *multi-rate max-min fairness*. They define an allocation to be multi-rate max-min fair if (1) it is feasible (i.e. no link is overloaded), (2) the multicast session is a multi-rate session, and (3) no receiver can increase its allocated rate without causing a decrease in the rate to a more constrained receiver.

The main difference between our work and [3] is in the architecture proposed to implement this fairness criterion. The solution proposed in [3] relies on the friendly behavior of the end-systems, which apply some kind of congestion control scheme. In contrast, our approach is *unfriendliness proof*, i.e. a flow cannot possibly gain a greater share of bandwidth by misbehaving.

III. The MFBA architecture

The *max-min fairness* criterion described in the previous section for unicast flows is implemented by the well known fair queuing discipline. Most of the implementations of fair queuing [8], [9], [10], [11], [12], [13] maintain state for each flow in all nodes. However, maintaining per-flow state has a cost and raises scalability issues in high speed networks that can jeopardize the deployment of these techniques. Recently, several approaches have been proposed to approximate fair queuing while limiting per-flow state at the slower edge nodes of the network and removing it from core nodes (see CSFQ [4], RFQ [14], *T*UF [15] and SCALE [16]).

In this section, we propose a network architecture, *Multicast Fair Bandwidth Allocation* (MFBA), that implements the *anycast max-min fairness* criterion proposed in the previous section for multicast and unicast flows. MFBA is the result of extending previous work on core stateless fair queuing architectures to multicast.

A. Flow Labeling

To avoid maintaining per-flow state at each router, MFBA uses a distributed algorithm in which only edge routers maintain per-flow state, while core routers do not maintain per-flow state but instead utilize the information carried via a label in the header of every packet. The edge router based on per-flow information assigns this label and the labels are examined at each router along the path.

Following the above explanation, at the ingress router of the network each packet of a flow i is assigned a label L_i equal to:

$$L_i = r_i^{send} \cdot unif(0,1) \tag{3}$$

where r_i^{send} is the estimated sending rate of flow i and $unif(0,1)$ is a random variable uniformly distributed between 0 and 1. This random labeling, combined with the core dropping of Section III-B, leads to the desired allocation, as shown at the end of Section III-B.

For the estimation of the sending rate of flow i, r_i^{send}, we use the same exponential averaging formula as in [4]. The reason for using that estimation is that it allows to bound the excess service[1] received by a flow (see Theorem 1 of [4]).

Specifically, let t_i^k and l_i^k be the arrival time and length of the k^{th} packet of flow i. The estimated sending rate of flow i, r_i^{send}, is updated for every new packet in flow i as:

$$(r_i^{send})_{new} = (1 - e^{-(T_i^k/K)})\frac{l_i^k}{T_i^k} + e^{-(T_i^k/K)} \cdot (r_i^{send})_{old} \tag{4}$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a constant. Following the rationale discussed in [4], in this paper we set K equal to 100 ms.

B. Core Dropping

In case of congestion, packets at core routers are dropped depending on their label L_i according to the following algorithm:

if $L_i > L_{fair}$ **then**
  drop(packet)
else
  enque(packet)
end if

One key aspect of the above algorithm is the estimation of L_{fair} (the *fair label*) for each link. L_{fair} should be such that, in case of congestion, the rate of enqued packets, F, equaled the link's capacity C. For scalability reasons, L_{fair} should be estimated without storing per-flow information at the core nodes. Different solutions to the problem of estimating L_{fair} without per-flow state have been proposed in [4], [14], [16], [17]. The algorithm that we use in this paper is the one proposed in [4].

To compute L_{fair}, [4] keeps two aggregate variables: A, the estimated aggregated arrival rate, and F, the estimated rate of the accepted traffic. Then, L_{fair} is updated every K time units according to the following algorithm:

if $A \geq C$ **then** {link congested}
  $(L_{fair})_{new} = (L_{fair})_{old} \cdot C/F$
else {link uncongested}
  $(L_{fair})_{new}$ = largest L_i observed
end if

We now investigate the bandwidth distribution resulting from the flow labeling and core dropping explained above.

At the ingress of the network, the packets of a flow i are labeled according to a random variable uniformly distributed between 0 and the flow's estimated sending rate. The throughput experienced by this flow i at the first congested link it crosses is equal to:

$$r_i = r_i^{send} \cdot prob(L_i < L_{fair}) = L_{fair} \tag{5}$$

where L_{fair} is the link's *fair label*. The label of the non-dropped packets of flow i at that link is uniformly distributed between 0 and L_{fair}, where L_{fair} is equal to the flow's outgoing rate from the link.

[1] According to the desired bandwidth allocation, during a given time interval a flow is entitled to receive a specified amount of service data. We define any amount above this, the *excess service*.

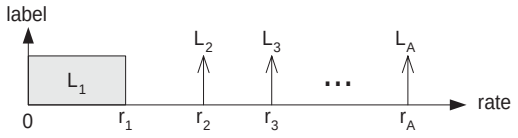

Fig. 1. Labeling for Layered Multicast.

Applying the above rationale recursively, it can be shown that the throughput obtained by any flow i contributing to congestion at a link l equals the link's *fair label*. As a consequence, two flows contributing to congestion at a link experience the same throughput in that link, which, according to Equation 2, leads to *anycast max-min fairness*.

IV. Layered Multicast Support

In MFBA, the bandwidth experienced by a receiver depends on the level of congestion of the links in the path from the sender to the receiver. In the case of multicast, this results in an heterogeneity on the amount of bandwidth experienced by the different receivers of the same flow. To cope with this heterogeneity, in this section we propose an alternative labeling scheme to the one proposed in Section III-A for the support of layered multicast within MFBA.

With layered multicast, we have that a multicast stream is divided in different layers such that layer 1 provides a minimum quality stream and each layer $a+1$ adds more quality to layer a. This feature allows to gracefully adapt the multicast flow's quality to the bandwidth available for each individual receiver, by dropping the higher layers until the resulting stream can be accomodated into the bandwidth available for the receiver.

In MFBA, in order to have the higher layers dropped first when the bandwidth to which a layered flow is entitled in a link is smaller than the flow's arrival rate, we assign higher labels to the higher layers, according to the following algorithm (see Figure 1 for an illustration). A packet of layer a is assigned a label L_a equal to:

$$L_a = \begin{cases} r_a^{send} \cdot unif(0,1), & a = 1 \\ r_a^{send} & a > 1 \end{cases} \quad (6)$$

where r_a^{send} is the aggregated sending rate of the layers lower or equal to a. Since at a link l packets with labels larger than the link's L_{fair} are dropped, the aggregate rate of the non-dropped layers at that link will be such that

$$r_a^{send} \leq L_{fair} \quad (7)$$

where L_{fair} is precisely the throughput to which the flow is entitled at link l. Thus, the idea behind the labeling scheme proposed is to forward in a link as many layers as possible with the flow's fair share of bandwidth in that link, starting from the lowest layer.

Note that in the labeling scheme of Equation 6, labels for layers higher than 1 take deterministic values, instead of random values as in Section III-A. This is to force having all the packets of a layer either forwarded or dropped, instead of having them forwarded with a certain probability. As we learnt in [7], a layer partially delivered causes a fluctuation in the number of received layers that can lead to an undesirable "flickering" effect in the resulting video.

In contrast to the above, in Equation 6 packets of layer 1 are labeled with random values. This is because it is preferable to partially receive layer 1 than to receive no data at all[2].

For the estimation of the aggregated rates r_a^{send} we propose the following formula. Let t_a^k be the arrival time of the k^{th} packet of layer a, and $\hat{l}_a^k$ the sum of the lengths of the packets of layer lower or equal to a transmitted between packets $k-1$ (excluded) and k (included) of layer a. Then, r_a^{send} is updated for every new packet of layer a according to[3]

$$(r_a^{send})_{new} = (1 - e^{-(T_a^k/K)})\frac{\hat{l}_a^k}{T_a^k} + e^{-(T_a^k/K)} \cdot (r_a^{send})_{old} \quad (8)$$

where $T_a^k = t_a^k - t_a^{k-1}$.

A natural concern of the labeling scheme proposed for layered flows is whether a flow can take advantage of the freedom to assign different layers to its packets to get more than its fair share of bandwidth. Actually, if the aggregated rates r_a^{send} were computed as the sum of the individual rates of each layer, it can be shown that there are some pathological cases in which a flow could exploit the freedom to assign layers to obtain more than the fair share.

Theorem 1 (at the appendix) answers the above concern when the value of L_{fair} is held fixed in all links. This theorem gives a bound on the excess service received by a flow during any time interval when using the *layered labeling* scheme proposed in this section. This bound is independent of the packet arrival process, the layers assigned to the packets and the length of the time interval. It does depend on the maximal rate at which the packets of the flow can arrive at a router (limited for example by the speed of the flow's access links) and on the number of layers allowed.

By bounding the excess service, Theorem 1 shows that the asymptotic throughput received by a flow cannot exceed its fair share of bandwidth. Thus, flows can only exploit the system over short time scales, and they are limited to their fair share of bandwidth over long time scales. This result is very important, since it constitutes the basis for the unfriendliness proof feature of the proposed architecture.

The resulting algorithm from the flow labeling, layered labeling and core dropping explained in sections III and IV is illustrated in Figure 2.

V. Implementation

In this section we present an implementation of the MFBA architecture for a PC-based router running under the Linux

[2]If all layers are sent at a constant rate, the proposed labeling scheme completely eliminates the flickering effect, since the number of layers received is constant. If the sending rate of each layer is variable, the flickering effect cannot be completely eliminated, however it is reduced as compared to using random labels. In a series of unreported experiments, we verified that, with the variable rate codec of Section VI, the use of deterministic labels ($L_a = r_a^{send}$ for $a > 1$) significantly contributed to reduce the flickering effect perceived, as compared to using the equivalent random labels ($L_a = r_{a-1}^{send} + (r_a^{send} - r_{a-1}^{send}) \cdot unif(0,1)$).

[3]Note that with the rate estimations of Equation 8, the computational complexity of the labeling scheme proposed in this section and the one of Section III-A are similar: both schemes require to perform one exponential computation for every new packet.

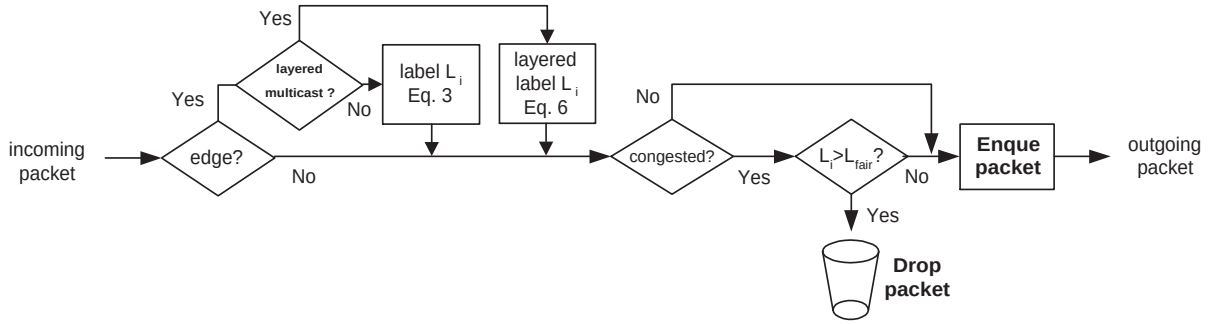

Fig. 2. Forwarding algorithm in MFBA

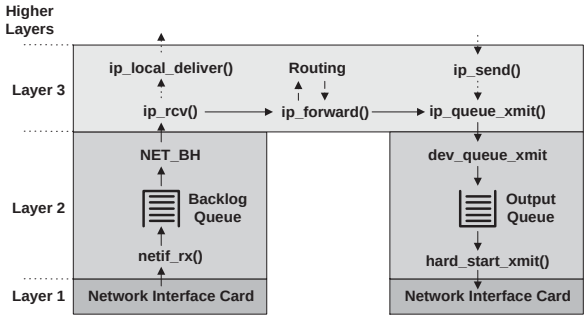

Fig. 3. The course of a packet through the system.

operating system. We describe the design and implementation issues of the different components of the architecture. This implementation was done with the 2.2.18 version of the Linux kernel.

The Linux operating system is a good basis for implementing a router with the MFBA functionality. It runs on standard PC hardware and source code of the kernel is widely available. In addition, it supports a variety of queuing disciplines for output queues of network devices, and all functions for routing are already provided.

To get a deeper insight into our concrete realization of the MFBA architecture, understanding how Linux handles basic network functions is important. Hence, a short overview of the standard Linux implementation is given next. Afterwards, the implementation of the MFBA architecture is presented.

A. Linux network implementation

To give a short overview of the Linux IP network implementation, the course of an IP-packet through the system is described first (see Figure 3).

After a packet is received by a network interface card, a hardware interrupt is triggered. As a consequence, an interrupt handling routine (named `ei_interrupt()`) is invoked and determines the type of interrupt. For interrupts caused by incoming packets a further handling routine is called (`ei_receive()`) which simply copies the packet from the network card into an internal *socket buffer* structure [18] and calls a procedure named `netif_rx()`. The latter queues the packet (represented by a socket buffer structure) into a central queue (backlog queue) consisting of all packets that arrived on any network adapter of the system. The first time-critical part of the interrupt routine, called 'top-half' is finished at this time.

The necessary second part, called 'bottom-half', is handled by the network bottom-half routine (`NET_BH`) which is regularly invoked by the kernel scheduler. At first, this procedure checks whether there are any packets waiting for transmission in any output queue of any network adapter. If there are any packets waiting they are processed for a limited period. Subsequently, `NET_BH` proceeds with the next packet of the backlog queue and determines the appropriate protocol to handle the packet which is in our case the Internet Protocol IP. `ip_rcv()` checks for correctness of the IP header and then processes any existing options. It also reassembles the original IP packet from fragments if necessary and if the packet has reached its final destination. In the latter case, the packet is delivered locally, otherwise it is routed and forwarded towards its destination. `ip_forward()` tries to find the right network adapter this packet is forwarded to next by use of a routing table. If there is a valid entry in the routing table `ip_queue_xmit()` is subsequently invoked, performing some final operations such as decrementing time-to-live values and recalculating IP header checksums. `dev_queue_xmit()` queues the packet into the output queue of the corresponding network device. At this point a special queueing discipline can be invoked. Thus, each queueing discipline constitutes one output queue for a device that is not necessarily served in a FIFO with Drop-Tail basis. Within a queueing discipline, transfer of a packet onto network media is initiated by calling `hard_start_xmit()`, which instructs the network device to send the packet.

The Linux kernel already contains various queueing disciplines apart from the standard FIFO, like Class Based Queueing, Weighted Fair Queueing or Random Early Detection to implement different network features like traffic control or differentiated services (see e.g. [19], [20]).

B. MFBA implementation

In our implementation of the MFBA architecture, we inserted the algorithm of Figure 2 in the queuing discipline of the output queue. This algorithm decides whether an incoming packet to the output queue is enqueued or dropped. This is illustrated in Figure 4.

The MFBA queueing discipline was implemented in a kernel module. Kernel modules need not to be present all the time in the kernel, so the kernel can run without them if they are not actually used. Particularly, instead of recompiling the

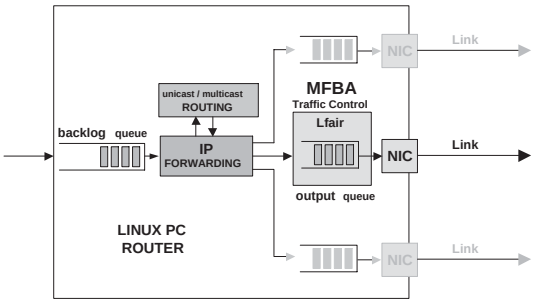

Fig. 4. MFBA implementation in Linux.

whole kernel and restarting the system every time a part of the module's code was changed, one can simply reload the newly coded module. This shortens development time drastically.

During the implementation of the MFBA queueing discipline, a number of issues had to be solved. In the following we provide a detailed description of the various implementation issues we faced.

1) Router performance: I/O performance and CPU router performance are crucial for successful operation, because if the PC is too slow, the protocol processing for a packet is not finished before a new packet arrives. As a consequence, the implemented MFBA algorithm for packet dropping is never used because packets are dropped already earlier in the backlog queue. Nevertheless, we checked that a PC with a AMD-K6 CPU running at 350 MHz (the machine we used as a router) is sufficient to route incoming traffic of 20 Mbps at least.

2) Router configuration: One of the parameters of the MFBA algorithm that needs to be configured is the capacity of the link C. In order to set this parameter, we measured the net capacity obtained in the 10 Mbps Ethernet link with a FIFO queue. The packet lengths considered to compute this capacity included the 42 bytes of overhead of the UDP, IP and Ethernet headers and the 4 bytes of the Ethernet checksum, in addition to the packet payload. Considering this overhead, the net capacity measured was of 9.8 Mbps; this is the value we used for C in the MFBA algorithm.

3) Label location in packet header: An important issue of the implementation is how to insert the label value L_k into the packet header. Two possibilities are: (1) introduce a new IP option, or (2) introduce a new header between layer 2 and layer 3, similar to the way labels are transported in Multi-Protocol Label Switching (MPLS). While both of these solutions are quite general and can potentially provide large space for encoding the label, for the purpose of our implementation we considered a third option: store the state in the IP header. By doing this, we avoid the penalty imposed by most IPv4 routers in processing the IP options, or the need of devising different solutions for different technologies as it would have been required by introducing a new header between layer 2 and layer 3.

The biggest problem of using the IP header is to find enough space to insert the label. The main challenge is to remain compatible with current standards and protocols. In particular, we want to be transparent to end-to-end protocols. One possibility to achieve this goal is to use the type of service (TOS) byte. However, as we discuss in the following subsection, the 8 bits obtained with this option are not sufficient to encode the label with the desired level of accuracy. Another option is to use *IP identifier* field, which has 16 bits. This field is unused for non-fragmented packets (fragmented packets are identified by the pair *more fragment* and *fragment offset*). As pointed out in [21], very few packets are actually fragmented in the Internet (0.22%). Therefore, we have chosen this latter option for the labeling in the MFBA architecture. Fragmented packets are ignored and forwarded as usual.

4) Label mapping: The next issue to solve was how to map the label values (which theoretically can be any real value) into the 16 bits of the *IP identifier* field. To represent a wide range of rates in the Internet while maximizing the accuracy, we used a logarithmic scale to map the labels between L_{min} and L_{max} to discrete integer values between 0 and $2^{16}-1$. Let L be the original label (real value) and V its integer representation in the 16 bit field. Then,

$$V = \left\lfloor (2^{16}-1)\frac{log_2(L) - log_2(L_{min})}{log_2(L_{max}) - log_2(L_{min})} \right\rfloor \qquad (9)$$

The above mapping had already been proposed in [15] but with log_{10} instead of log_2. The reason to use log_2 is because working in base 2 allows to perform operations more efficiently. Specifically, the computation of 2^x, which is required at core nodes to unmap label values, can be very easily performed by shifting x positions the bit representation of 1.

With Equation 9, we can map labels between 1 (L_{min}) and 2^{32} (L_{max}) with an error bounded by 0.04%. Note that, using the 8 bits of the TOS field instead of the 16 bits of the *IP identifier* field, this error bound is of 9.09%, which is unacceptably high.

5) Detection of layered flows at ingress nodes: As explained in previous sections, the MFBA architecture uses two different labeling schemes: one for multicast layered flows (Eq. 6) and one for any other type of flow (Eq. 3). Therefore, at the ingress node, one of the two labeling schemes has to be selected for every incoming packet.

In our implementation we have used multicast layered transmissions running RTP on top of UDP/IP. In order to select the appropriate labeling scheme we detect packets belonging to multicast layered flows by examining the *payload type* field within the RTP header of every packet.

Further, for packets belonging to layered flows, the layer within a flow that a packet belongs to must be determined in order to use the labeling scheme of Eq. 6. In our implementation, every packet carries its layer information in an RTP header extension, which we already defined in [7]. There, the layer information is encoded in a 6-bit field, which limits the maximum number of layers allowed to 64.

6) Rate estimation at core nodes: The last issue we had to solve for the implementation was the rate estimation. For the estimation of A and F at core nodes we decided to use the *Time Sliding Window* (TSW) algorithm [22] (Equation 10) instead of the exponential averaging of Equation 4. The reason why we chose the TSW algorithm was to avoid expensive exponential computations at core nodes. The experimental results show that this change does not affect the accuracy of

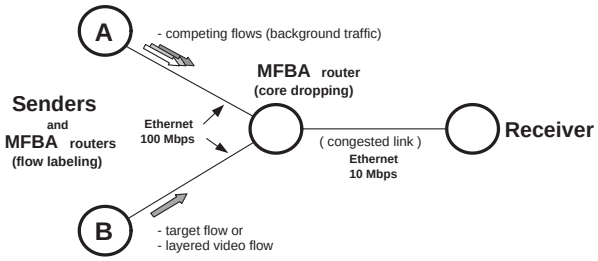

Fig. 5. Configuration of the test network.

the estimation of L_{fair}.

$$r_{new} = \frac{l_k}{T_k + K} + \frac{K}{T_k + K} \cdot r_{old} \tag{10}$$

VI. Experimental Results

To evaluate the performance of the MFBA architecture, we performed some tests with the implementation explained in the previous section.

We first performed some tests on the bandwidth allocation for constant and variable traffic (Sections VI-A and VI-B). The goal of this first set of tests was to validate the implementation of the architecture and understand its bandwidth sharing behavior.

The second set of tests focused on layered multicast (Sections VI-C and VI-D). The goal was to analyze the performance in terms of allocated bandwidth and resulting stream quality of the layered multicast extension proposed. For this purpose, we used a layered video flow.

For all tests, we used the following configuration that is shown in Figure 5. The testbed comprised a PC as MFBA router (AMD-K6 CPU at 350 MHz), two PCs as sender/edge (Pentium CPU at 200 MHz) and one PC as receiver (AMD-K6 CPU at 300 MHz). The network was build of separate Ethernet segments (two 100 Mbps and one 10 Mbps). The latter segment, that connected the router with the receiver, was the bottleneck link.

We used the following parameters for the tests. Packets corresponding to constant and variable traffic had a constant UDP payload length of 1000 bytes. In addition to this payload, the packet length value used to compute rates (Equation 4) also included the 42 bytes of overhead of the UDP, IP and Ethernet headers. All tests were executed for a fixed duration of 120 s.

A. Constant traffic

With the *anycast max-min fairness* criterion, the bandwidth allocated to a (unicast or multicast) flow i, r_i, can be expressed as

$$r_i = min(r_i^{send}, L_{fair}) \tag{11}$$

In order to validate that the above bandwidth allocation was complied, we performed the following test. We congested the bottleneck link of the testbed with 9 background flows from sender A, each flow sending at a constant rate. The aggregated sending rate of the 9 background flows was equal to the link's capacity.

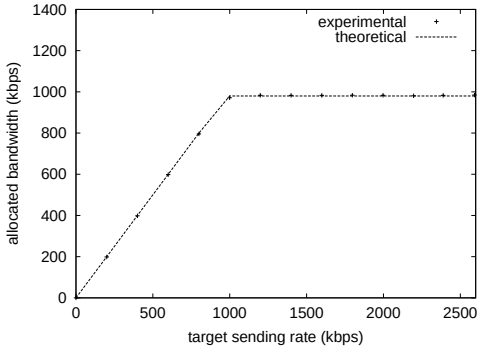

Fig. 6. Bandwidth Allocation with constant traffic.

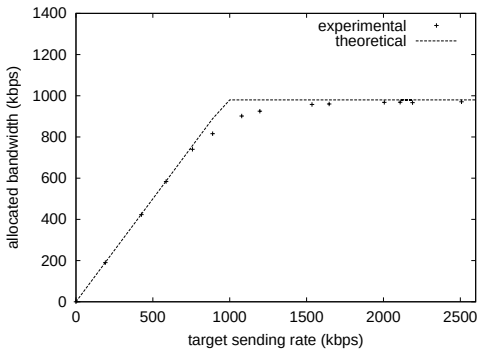

Fig. 7. Bandwidth Allocation with variable traffic.

Sender B sent an additional flow, the target flow, also at a constant rate. The sending rate of this flow ranged from 0 to 2.5 Mbps.

Figure 6 shows the results obtained for the above test (throughput obtained by the target flow). It can be observed that the results obtained are surprisingly accurate: the actual allocated rate matches exactly the theoretical rate (Equation 11).

We conclude that the MFBA architecture provides the desired bandwidth allocation for constant traffic.

B. Variable traffic

Bandwidth allocation is much easier when dealing with constant traffic than when dealing with variable traffic. To study the behavior of the MFBA architecture in the latter case, we repeated the previous experiment but with the target flow sending at a variable rate instead of at a constant rate.

Specifically, the variable flow consisted of the aggregation of 10 ON/OFF sources, with idle times exponentially distributed (average 500 ms) and active times Pareto distributed (average 500 ms and *shape* equal to 1.3). The average sending rate of the variable flow ranged from 0 to approximately 2.5 Mbps.

Figure 7 shows the results corresponding to the above test. From these results it can be observed that for an average sending rate around the fair share (0.98 Mbps), the variable flow receives a lower throughput than its fair share. When the sending rate increases, then the flow's throughput tends asymptotically to the fair bandwidth share.

The explanation for the above behavior can be found in the time plots given in Figures 8 and 9. These time plots cor-

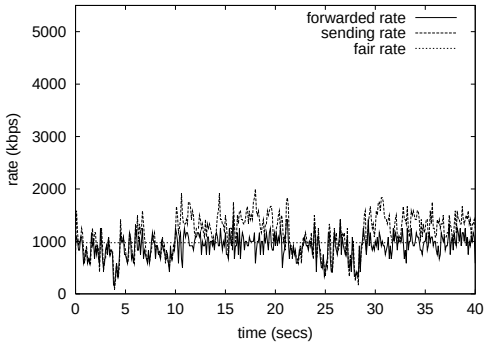

Fig. 8. Instantaneous Bandwidth Allocation with variable traffic.

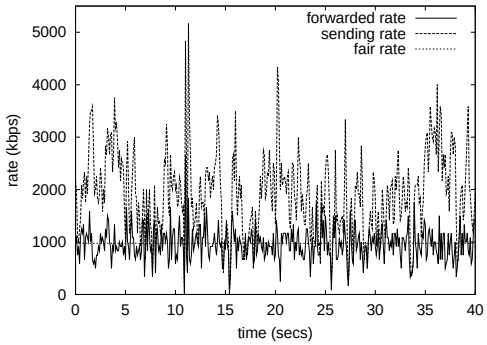

Fig. 9. Instantaneous Bandwidth Allocation with variable traffic.

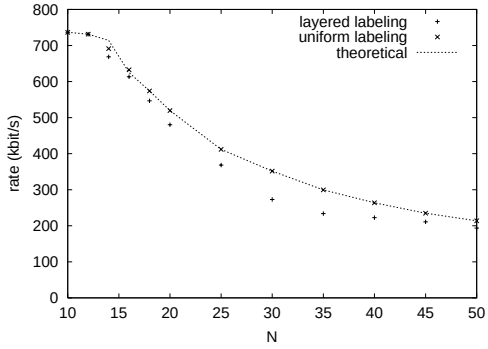

Fig. 10. Bandwidth Allocation of the layered video flow.

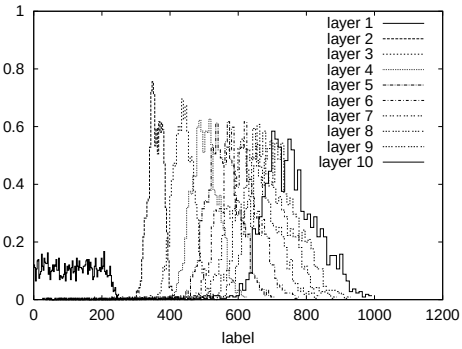

Fig. 11. Per-layer label histogram of the layered video flow.

respond to average sending rates of 1 Mbps and 2 Mbps respectively. For an average sending rate of 1 Mbps, the instantaneous sending rate of the variable flow fluctuates between values below and above the fair share. When the instantaneous rate is below the fair share, the forwarded rate equals to the sending rate. When it is above, then the forwarded rate equals the fair share. As a result, the throughput obtained by the variable flow is below the fair share.

In contrast to the above, when the flow's average sending rate is 2 Mbps, most of the time the flow is sending at a rate above the fair share, which results in a forwarded rate equal to the flow's fair share. As a consequence, the throughput obtained by the flow in this case is much closer to the flow's fair share of bandwidth.

C. Layered Video with constant background traffic

In Section IV we have proposed a labeling scheme for the transmission of layered multicast flows that adapts the quality of the delivered stream to the bandwidth available for each receiver. The design goal was to find a labeling scheme that led to a graceful degradation of the flow's quality as the available bandwidth decreased while preserving the original anycast max-min bandwidth allocation.

In order to evaluate the performance of the proposed approach we ran some experiments with layered video. We transmitted a layered video flow and compared the video quality at the receiver when using *layered* labeling (Equation 6) and *uniform* labeling (Equation 3) for different bandwidth allocations. Experiments were done using the DCT-based layered video codec described in [7]. The original video was coded using 10 layers. To quantify the level of corruption of the received videos due to packet losses, we used the QMeasure metric described in [23]. This measure tries to quantify the distortion perceived by the human visual system and is known to be more meaningful than objective measures like the PSNR or the MSE.

In the first test we performed, Sender B transmitted the layered video flow, while Sender A sent a background traffic composed of $N-1$ flows, each sending at a constant rate. The background traffic congested the bottleneck link with a load corresponding to 120% of the link's capacity.

Figure 10 shows the resulting bandwidth allocation with *uniform* and *layered* labeling. This result validates the labeling scheme we proposed in Equation 6. With uniform labeling, the flow's bandwidth is approximately the flow's fair share of bandwidth. With layered labeling it is slightly lower, due to the fact that the last layer is totally discarded. Note that, for $N \geq 30$, the only layer remaining is the first one.

Figure 12 shows how the bandwidth of the video flow is distributed among the different layers with both labeling schemes when $N = 20$. It can be observed that the layered labeling scheme achieves a good level of discrimination among layers, as desired. However, this discrimination is not perfect, due to the fact that the sending rate of each layer is variable (this variability is illustrated in the per-layer label histogram of Figure 11). Note that with uniform labeling, there is no discrimination since the labeling does not distinguish among layers.

Figure 13 shows the benefit obtained with layered labeling as compared to uniform labeling. In this figure, video quality is measured with the QMeasure metric mentioned above. This metric is such that the lower the QMeasure, the higher the video quality. We can observe that, with *layered* labeling, the perceived quality is degraded gracefully (i.e. the QMea-

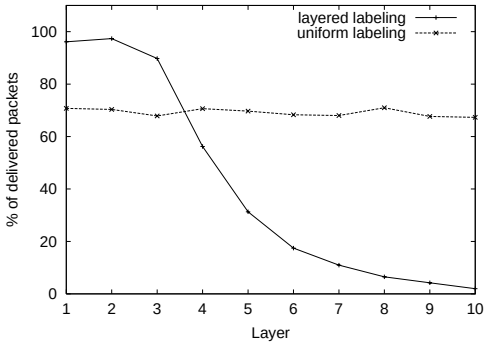

Fig. 12. Percentage of delivered packets of every layer of the video flow.

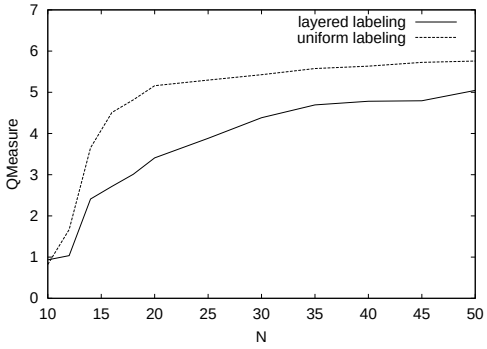

Fig. 13. Perceived video quality with constant background traffic.

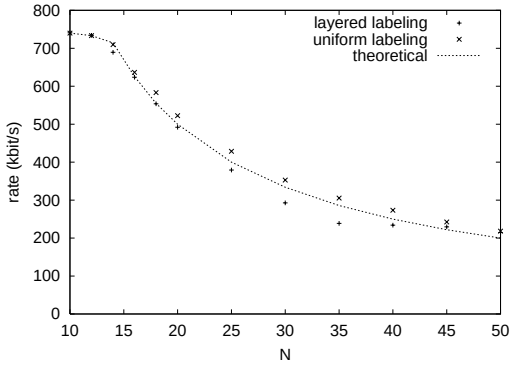

Fig. 15. Bandwidth Allocation of the layered video flow.

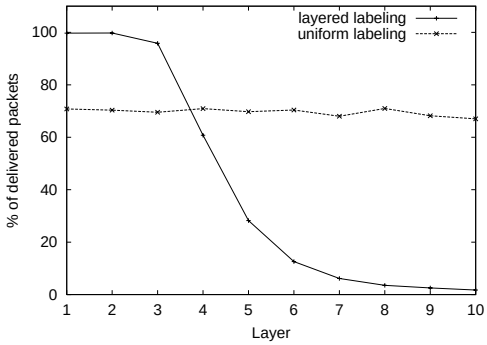

Fig. 16. Percentage of delivered packets of every layer of the video flow.

sure keeps lower) as the number of competing flows (N) increases. In contrast, with *uniform* labeling the quality suffers a sharp degradation when the bandwidth allocated to the multicast layered flow becomes smaller than the flow's sending rate.

The results reported by the QMeasure match our subjective perception of the videos. With the *uniform* labeling, when reducing the allocated bandwidth, we observed a jerky display where the movement of objects was corrupted (because of the loss of entire frames), overlapping of images and frequent changes of definition within the same frame. In contrast, with the *layered* labeling, we only observed a smooth decrease in the definition. These effects are shown in the snapshots of Figure 14.

D. Layered Video with variable background traffic

In order to analyze the performance of the proposed approach for layered multicast when the background traffic is variable, we repeated the previous experiment with variable background traffic instead of constant. Specifically, the $N-1$ background flows were variable flows of the type described in Section VI-B.

Figure 15 shows the resulting bandwidth allocation in the above test. It can be observed that, with a variable background traffic, the video stream receives a larger throghput than with a constant background traffic. The reason is that, as we have seen in Section VI-B, a variable flow consumes a throughput lower than its fair share. Similarly, a variable background traffic consumes less bandwidth than a constant one, and, as a result, there is more bandwidth left for the video flow.

Figure 16 shows how the bandwidth of the video flow is distributed among the different layers when $N = 20$. It can be observed that level of discrimination achieved is almost identical to the one we had with a constant background traffic. That is, with the level of burstiness of the variable background traffic considered in this section, the resulting fluctuation over time of the fair bandwidth share has a negligible impact into the layer discrimination obtained.

Finally, Figure 17 shows the resulting video quality. The behavior is similar than with a constant background traffic: With *layered* labeling, the perceived quality is degraded gracefully as the number of competing flows (N) increases, while with *uniform* labeling the quality suffers a sharp degradation when the bandwidth allocated to the multicast layered flow becomes smaller than the flow's sending rate.

We conclude that the results obtained in Sections VI-C and VI-D validate the approach proposed in this paper for

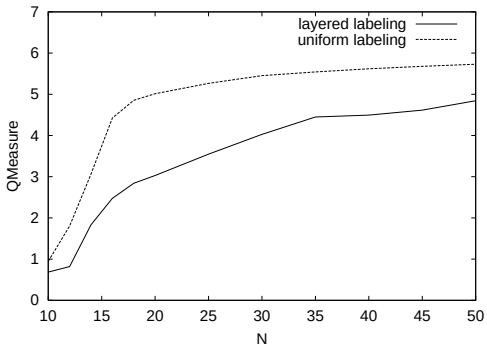

Fig. 17. Perceived video quality with variable background traffic.

Fig. 14. Snapshots of the video experiments. The leftmost image corresponds to a frame of the original video. The image in the center shows how the same frame was received with *uniform labeling* and the rightmost when using *layered labeling*.

layered multicast: both for constant and for variable background traffic, our scheme achieves the goal of having a smooth quality degradation as the bandwidth available for a given receiver decreases.

VII. Summary and Conclusions

The *anycast max-min fairness* criterion results from extending to multicast the well known max-min fairness criterion for unicast.

In this paper we have proposed an architecture to efficiently implement the *anycast max-min fairness* criterion: the MFBA architecture. MFBA is inspired on previous work to avoid maintaining per-flow state at core nodes. As a result, MFBA scales well with the number of flows.

MFBA has been designed to allow a multicast layered flow to accommodate its quality to the available bandwidth of each receiver. The main features of the layered multicast support in MFBA are that it requires neither signaling nor the interaction of the receiver, it reacts immediately upon changing network conditions and it is unfriendliness proof.

The proposed approach for layered multicast has been studied both analytically and experimentally.

Via analytical studies we have shown that a user cannot possibly gain a greater share of bandwidth than the flow's fair share by maliciously assigning the flow's packets to different layers (that is the basis of the unfriendliness proof feature).

The benefit of using layered multicast in terms of the quality experienced by a receiver has been studied by using our Linux implementation of the MFBA architecture and a layered video codec. Experimental results have shown that, with the proposed layered multicast scheme, the resulting video quality is gracefully degraded as the bandwidth allocated for a receiver decreases, both for constant and for variable background traffics.

References

[1] J. Bolot, T. Turletti, and I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet," in *Proceedings of ACM SIGCOMM '94*, London, UK, August 1994, pp. 58–67.

[2] S. McCanne and V. Jacobson, "Receiver-driven layered multicast," in *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[3] D. Rubenstein, J. Kurose, and D. Towsley, "The Impact of Multicast Layering on Network Fairness," in *Proceedings of ACM SIGCOMM '99*, Boston, MA, September 1999.

[4] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, August 1998, pp. 118–130.

[5] T. Kim, R. Sivakumar, K.-W. Lee, and V. Bharghavan, "Multicast Service Differentiation in Core-Stateless Networks," in *Proceedings of International Workshop on Networked Group Communication*, Pisa, Italy, November 1999.

[6] A. Flores and M. Ghanbari, "Prioritised delivery of layered coded video over IP networks," *ACM Transactions on Multimedia*, 2001.

[7] F. Raspall, C. Kuhmuench, A. Banchs, F. Pelizza, and S. Sallent, "Study of packet dropping policies on layered video," in *Proceedings of Packet Video Workhsop*, Korea, April 2001.

[8] J. C. R. Bennet and H. Zhang, "Wf2q: Worst-case Fair Weighted Fair Queuing," in *Proceedings of IEEE INFOCOM '96*, San Francisco, CA, March 1996.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," in *Proceedings of ACM SIGCOMM '95*, Cambridge, MA, September 1995.

[10] P. Goyal, H. M. Vin, and H. Cheng, "Start-time Fair Queuing: a Sscheduling Algorithm for Integrated Serivces Switched Networks," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 690–704, October 1997.

[11] S. Keshav, "On the Efficient Implementation of Fair Queuing," *Journal of internetworking: Research and Experience*, vol. 2, pp. 57–73, September 1991.

[12] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single-node Case," *IEEE/ACM Transactions on Networking*, vol. 1, June 1993.

[13] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," in *Proceedings of ACM SIGCOMM '95*, Cambridge, MA, September 1995.

[14] Z. Cao, Z. Wang, and E. Zegura, "Rainbow Fair Queueing: Fair Banwdith Sharing Without Per-Flow State," in *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.

[15] A. Clerget and W. Dabbous, "*T*UF: Tag-based Unified Fairness," in *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[16] H. Zhu, A. Sang, and S. Li, "Weighted Fair Bandwidth Sharing Using SCALE Technique," *Computer Communications Journal, Special Issue in QoS*, vol. 24, no. 1, January 2001.

[17] M. Nabeshima, T. Shimizu, and I. Yamasaki, "Fair Queuing with In/Out Bit in Core Stateless Networks," in *Proceedings of the Eight IEEE/IFIP International Workshop on Quality of Service (IWQoS 2000)*, Pittsburg, PA, June 2000.

[18] A. Cox, "Network buffers and memory management," *Linux journal*, September 1996.

[19] W. Almesberger, "Traffic Control implementation overview," `ftp://lrcftp.epfl.ch/pub/people/almesber/tcio-current.ps.gz`.

[20] K. Wehrle R. Bless, "Evaluation of differentiated services using an implementation under linux," in *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99)*, London, England, May 1999.

[21] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management," in *Proceedings of ACM SIGCOMM '99*, Boston, MA, September 1999, pp. 81–94.

[22] D. D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Services," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, August 1998.

[23] C. Kumuench, G. Kuehne, C. Shremmer, and T. Haenselmann, "A video-scaling algorithm based on human perception for spatio-temporal stimuli," in *Proceedings of SPIE, Multimedia Computing and Networking*, 2001.

APPENDIX

Theorem 1: Consider a link with a constant fair label L_{fair} and a layered flow crossing this link. Then, the excess service received by this flow, that sends at a rate no larger than C (the capacity of the access link) is bounded above by

$$(A+1)l_{max} + L_{fair} \cdot K(A+1+2ln\frac{C}{L_{fair}}) \quad (12)$$

where A is the maximum number of layers allowed and l_{max} represents the maximum length of the packet.

Proof: Without loss of generality let us assume that exactly n packets are sent during the target interval I. Since, according to the dropping algorithm of Section III-B packets with labels above L_{fair} are dropped, the total service received by the flow in the interval I is

$$F = \sum_{i=1}^{n} l_i \cdot 1_{L_i \leq L_{fair}} \quad (13)$$

where l_i is the length of the i^{th} packet of interval I and L_i is its label.

Thus, the problem can be reformulated as to maximize F subject to the labeling of Equation 6. We first study which labels for packets of layer A maximize F.

It can be easily seen that the service received by a flow is maximized when: a) all the packets of the layer A are accepted, or b) there exists a time after which all packets of layer A are dropped. The reason is because having a packet with a label above L_{fair} provides no service at present, since the packet is dropped, and reduces the amount of service that can be obtained in the future for layer A, since it makes larger the current estimated aggregated rate of the layer. In the following we only consider the packets of layer A accepted.

Let I_{A1} be the subinterval of I before the arrival of the first packet of layer A, I_{A2} the subinterval comprised between the first and the last packet of layer A and I_{A3} the subinterval after the last packet of layer A. Let t_A^k be the arrival time of the k^{th} packet of layer A, and $\hat{l}_A^k$ the sum of the lengths of the packets of layer lower or equal to A transmitted between packets $k-1$ (excluded) and k (included) of layer A. Let A_n be the number of packets of layer A in the interval and A_1 the first packet of layer A. Then, F can be expressed as

$$F = \sum_{i \in I_{A1}} l_i + l_{A_1} + \sum_{k=2}^{A_n} \hat{l}_A^k + \sum_{i \in I_{A3}} l_i \quad (14)$$

From Equation 8 it can be derived

$$\hat{l}_A^k = \frac{r_A^k - r_A^{k-1} e^{-(T_A^k/K)}}{1 - e^{-(T_A^k/K)}} T_A^k \quad (15)$$

where $T_A^k = t_A^k - t_A^{k-1}$.

Substituting the above into Equation 14, we can express F as a function of r_A^k ($1 \leq k \leq A_n$). Then,

$$\frac{\partial F}{\partial r_A^k} = \frac{T_A^k}{1 - e^{-(T_A^k/K)}} - \frac{T_A^{k+1} e^{-(T_A^{k+1}/K)}}{1 - e^{-(T_A^{k+1}/K)}}, \quad 2 \leq k < A_n \quad (16)$$

and

$$\frac{\partial F}{\partial r_A^{A_n}} = \frac{T_A^{A_n}}{1 - e^{-(T_A^{A_n}/K)}} \quad (17)$$

Since $x/(1-e^{-x}) \geq 1$ and $xe^{-x}/(1-e^{-x}) \leq 1$ for any $x \geq 0$, we have

$$\frac{\partial F}{\partial r_A^k} \geq 0, \quad 2 \leq k \leq A_n \quad (18)$$

Consequently, F is maximized when r_A^k achieves its maximum value for $2 \leq k \leq A_n$. Considering that $L_A^k = r_A^k$ and $L_A^k \leq L_{fair}$, this value is $r_A^k = L_{fair}$. Substituting this into Equation 15 leads to

$$\hat{l}_A^k = \frac{L_{fair} - L_{fair} e^{-(T_A^k/K)}}{1 - e^{-(T_A^k/K)}} T_A^k = L_{fair} T_A^k, \quad 2 < k \leq A_n \quad (19)$$

and, assuming $T_A^k \ll K$,

$$\hat{l}_A^k \leq \frac{L_{fair}}{1 - e^{-(T_A^k/K)}} T_A^k \approx L_{fair} K, \quad k = 2 \quad (20)$$

Bounding l_{A_1} by l_{max} and substituting the above results into Equation 13 leads to the following expression for F:

$$\begin{aligned} F &= \sum_{i \in I_{A1}} l_i + l_{max} + KL_{fair} + L_{fair} \sum_{k=3}^{A_n} T_A^k + \sum_{i \in I_{A3}} l_i \quad (21) \\ &< \sum_{i \in I_{A1}} l_i + l_{max} + KL_{fair} + L_{fair} T_{A2} + \sum_{i \in I_{A3}} l_i \end{aligned}$$

where T_{A2} is the length of the time interval I_{A2}.

Using an identical argument as the one used above for layer A for layers $A-1$, $A-2$, ..., 2 it can be easily shown that the maximum service received by the flow is bounded above by

$$F < \sum_{i \in I_1} l_i + (A-1)l_{max} + (A-1)KL_{fair} + L_{fair} T_2 + \sum_{i \in I_3} l_i \quad (22)$$

where I_1 is the subinterval of I before the arrival of the first packet with layer higher than 1, I_2 the subinterval comprised between the first and the last packet with layers higher than 1 and I_3 the subinterval after the last packet with layer higher than 1.

The service that the flow can obtain sending only layer 1 during a time interval T_{l1} is bounded above by Theorem 1 of [4]:

$$F_{l1} < l_{max} + KL_{fair} + KL_{fair} ln\frac{C}{L_{fair}} + L_{fair} T_{l1} \quad (23)$$

Applying the above bound to intervals I_1 and I_3 and combining this with Equation 22 leads to

$$F < (A+1)l_{max} + L_{fair} \cdot K(A+1+2ln\frac{C}{L_{fair}}) + L_{fair} \cdot T \quad (24)$$

where T is the length of interval I.

Since $L_{fair} \cdot T$ is the service to which the flow is entitled during interval I, the proof follows. ■