

Extended Expedited Forwarding: In-Time PHB

Ulf A. Bodin * Andrej Brodnik †* Johan Karlsson * Andreas Nilsson *

Abstract

Today real-time applications are becoming common applications in the Internet. Such applications can benefit from guarantees on packet delays and loss-rate. We propose a new per-hop behavior called the IT PHB which provides such guarantees, to prioritized application flows, and allow excess traffic which is forwarded in-order in respect to the conforming traffic. Our proposed solutions, which only needs $O(1)$ amortized time for enqueueing and dequeueing, are evaluated using a network simulator.

1 Introduction

The use of real-time applications in the Internet is becoming increasingly common, e.g. video and IP telephony. Such applications need to present data to users as fast as possible. Consequently, most real-time applications are delay-sensitive. Real-time applications may also prefer low loss-rates (e.g. to present high quality sound and/or video to users). Applications requiring very low loss-rates are said to be intolerant (in contrast to applications that are tolerant to packet loss).

Clearly, delay-sensitive and intolerant applications gain from guarantees on bounded delay and loss. Such guarantees can be provided to prioritized traffic aggregates with the Expedited Forwarding (EF) per-hop behavior (PHB) [5]. EF requires that the amount of traffic using the PHB is limited. Prioritized traffic entering the networked is divided into conforming traffic and excess traffic. Excess traffic is dropped and conforming traffic (which is below the bit-rate allocated for EF in the network) is given loss-free forwarding with low queuing delay. This approach fits applications that need a certain bit-rate, but do not gain from more forwarding capacity if available.

Delay-sensitive and intolerant applications that can gain from additional forwarding capacity are not well supported by the EF approach. Such applications can adapt their sending rate as response to packet loss (i.e. they are rate-adaptive). Still, they may need loss-free forwarding of specific packets up to a certain bit-rate. To meet the needs of delay-sensitive, intolerant and rate-adaptive applications, an approach of differentiating traffic that allow excess traffic to be sent into the network is needed.

Most real-time applications require data to be ordered before processing. For such applications, data need to be ordered at the receiver before processing if packets are re-ordered by the network. This introduces delay. Hence, it is important to not re-order conforming and excess packets within an application data stream.

In this paper a new PHB group is defined, the In-Time (IT) PHB group. The IT PHB group provides delay limited and in-order forwarding of conforming and excess traffic aggregates. In addition, loss-free forwarding up to a specified bit-rate is guaranteed to conforming traffic. Excess traffic is given a loss-rate equal or close to the loss-rate given to best-effort traffic. Hence, excess traffic and best-effort traffic competes for available forwarding capacity.

EF is quite straightforward to implement in routers. A prioritized queue in addition to the best-effort queue is sufficient to support this PHB. IT, on the other hand, requires more sophisticated packet scheduling. Unfortunately, sophisticated scheduling mechanisms tend to consume considerably more clock cycles and memory than simpler scheduling mechanisms such as prioritized queuing. We show however that a scheduling mechanism supporting IT can be implemented efficiently. First, a naive implementation using a simple extension to the prioritized queue solution is presented. Finally, a more sophisticated scheduling mechanism that uses local queue tickets is presented.

The extended prioritized queue and the queue ticket based scheduling mechanism have one configurable parameter only, i.e. the maximum allowed delay for conforming and excess traffic. The amount of excess traffic that is forwarded increases with this delay limit. We show, through simulations with the network simulator version 2 (NS-2) [6], that IT can be provided but a more extensive evaluation is needed.

The problem of supporting IT can be formalized as

Definition 1 *Given a maximum allowed delay for prioritized packets, maintain a set of packets from both prioritized and best-effort flows such that the following operations are supported:*

- **enqueue(Packet *p)**
Add a new packet to to the set. (All packets have a tag indicating if it is a conforming, excess or best-effort packet). If packets need to be dropped excess and best-effort packet should be dropped fairly while conforming packets should never be dropped.
- **Packet *dequeue()**
Return the next packet to be sent, such that prioritized packets are sent in time and in-order. More-

*Department of Computer Science and Electrical Engineering, Luleå University of Technology, Luleå, Sweden

†Department of Theoretical Computer Science, Institute of Mathematics, Physics, and Mechanics, Ljubljana, Slovenia

over, best-effort and excess packets should be sent with close to the same ratio as they are enqueued. Excess packets which are late or out of order in respect to conforming packets should not be sent but dropped.

1.1 Notations and Support

We will be using FIFO buffers with bounded maximum size in the solutions to EF and IT. These buffers support the following operations:

- `enqueue(Item i)`
Add `i` to the end of the buffer if `size() < max()` otherwise do nothing.
- `Item first()`
If not empty return the first item in the buffer. If the buffer is empty a default item is returned.
- `dequeue()`
Remove the first item in the buffer.
- `int size()`
Return the number of items in the buffer.

We use N_b to denote the maximum number of items in buffer b . We assume that the FIFO buffers can be implemented to support the above operations in $O(1)$ worst case time using $O(N_b \cdot \text{sizeof}(Item))$ space.

We will use t to denote the current time and d to denote the maximum allowed delay.

1.2 Paper organization

In the following section we describe the different solutions of IT and their time and space requirements. In Sect. 3 we present the NS simulation and the functional testings of the proposed solutions. Finally, in Sect. 4 we conclude the paper and discuss some future improvements.

2 Solutions

First we describe how EF can be implemented using a prioritized queue as suggested by Jacobson et al. [5]. Then we will see how this solution can be extended to support IT. Finally we propose, in our opinion, a better solution to support the IT PHB.

2.1 EF implemented using a prioritized queue

Expedited forwarding can, as suggested by Jacobson et al., be implemented using a prioritized queue. We choose to use two FIFO buffers to implement the EF PHB; one buffer for the prioritized packets and one for the best-effort packets. Whenever there is a packet in the prioritized buffer we dequeue the first of them and if there are no prioritized packets we dequeue the first best-effort packet.

Note that the best-effort traffic will be starved if too much prioritized traffic arrive to the node. However, we assume that either the users are fair or that there is some sort of policing mechanism at the ingress point. Also note that packets will be dropped if they can not fit in the buffer which for practical reasons is bounded in size.

Obviously, both enqueueing and dequeueing can be done in $O(1)$ time. The space used, in addition to the space used to store the packets, is the space needed in the buffers to store pointers to the packets. Hence $M_c = N_c$ and $M_b = N_b$ denote the space for the conforming and best-effort buffer respectively. We will use *EF* to denote this solution.

2.2 IT extension to the prioritized queue solution

It is clear that, if excess traffic is allowed, the suggested EF implementation will not support IT, since it will not be able to dequeue the excess traffic in order with respect to the conforming traffic without treating it as a conforming traffic.

Hence, we extend the EF solution to meet our demands in the most natural way; we store the excess packets in the buffer for the best-effort packets. We also store, with a packet its sequence number σ and a time stamp $\tau = t + d$, to know the order of arrival to the node and when a prioritized (conforming/excess) packet has to leave the node at the latest.

Since the in-link bit-rate might be higher than the out-link bit-rate, two packets can arrive so close in time that it is impossible to send both of them in-time if the first is delayed as long as possible, which can be seen in Figure 1. Hence we need to change the time stamps of previous packets to ensure that the just arrived packet will be sent in-time. However, since we might need to adjust the time stamp of all the packets in the buffer we can not afford to do this. Instead we maintain a time offset, Δ , for the clock and whenever we detect that two packets arrive too close we add the needed time, $\delta = (t_2 + d) - ((t_1 + d) + \text{sendTime}(p_1))$, to the offset Δ . This offset is then added to the current time whenever deciding which type of packet should be sent (see details below). We also need to store the added time, δ , together with the packet in order to be able to decrease the offset when the packet is sent. We only do this for conforming packets since we are allowed to drop the excess packets if they are late.

A similar problem can occur if we have several in-interfaces into the node, in which case two packets can arrive on different interfaces at the same time. However, we assume that all packets arrive to the out-interface through a common back-plane in the node and hence this will not effect the out-interface.

We will use subscript ρ to denote that τ_ρ , o_ρ and σ_ρ is the time stamp, time offset and sequence number for the first packet p_ρ in the buffer ρ , respectively. Hence ρ is either c , e or b for the conforming, excess and best-effort packets respectively. Finally, we need to keep track

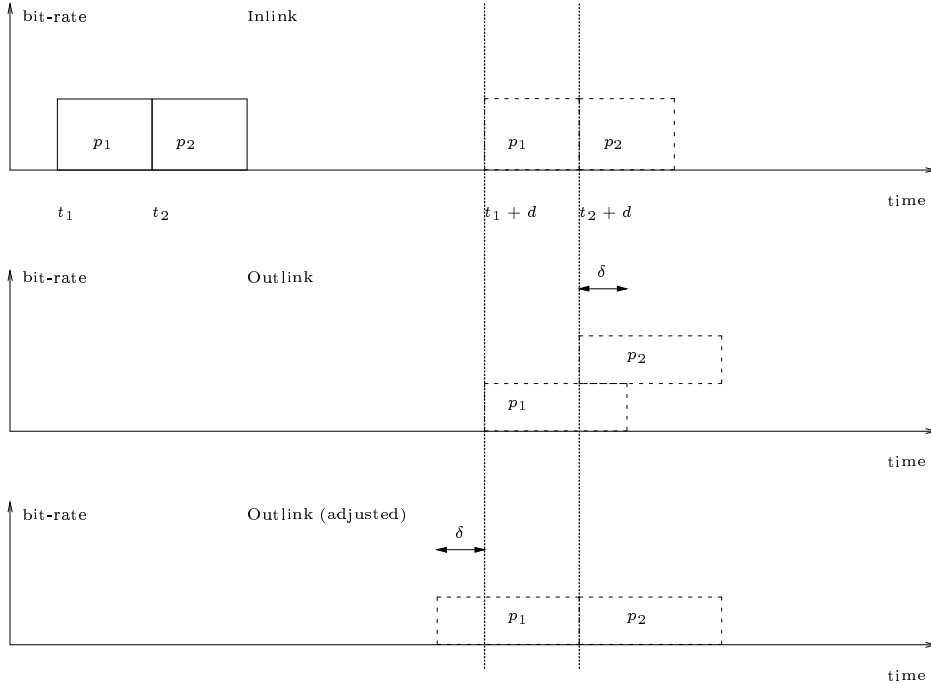


Figure 1: If p_1 is delayed until $t_1 + d$, then p_2 can not be sent before $t_2 + d$. Hence p_1 must be sent at $t_1 + d - \delta$.

of the sequence number for the last dequeued conforming packet Σ . Hence, whenever dequeuing a conforming packet we also store its sequence number.

To decide from which buffer a packet should be dequeued we first note that an excess packet should be dropped if it is late, $\tau_e < t$, or has a sequence number lower than that of the last send conforming packet, $\sigma_e < \Sigma$. Now we have the following steps:

1. If both buffers are empty a predefined value is returned.
2. If the best-effort/excess buffer is empty dequeue from the conforming buffer.
3. If the conforming buffer is empty dequeue the first packet from the best-effort/excess buffer unless it is an excess packet which should be dropped, in which case we drop it and start over from 1.
4. Check whether a best-effort or an excess packet is first in the best-effort/excess buffer.
 - If it is a best-effort packet dequeue it if it can be sent before the deadline of the first confirming packet ($(t+O)+sendTime(p_b) < (\tau_c+o_c)$), otherwise dequeue the conforming packet.
 - If it is an excess packet, first check if it should be dropped. If so drop it and start over from 2, otherwise check if the conforming packet should be sent before (due to sequence number ($\sigma_c < \sigma_e$) or time ($(t+O)+sendTime(p_e) > (\tau_c+o_c)$)). If so dequeue the conforming packet. Otherwise dequeue the excess packet.

This solution, which we denote *naive*, needs $4N_c + 3N_{eb} + O(1)$ memory compared to $N_c + N_b + O(1)$ for the

EF solution in the buffers. The enqueueing is still done in worst case constant time, but the dequeuing may need to drop excess packets. In worst case the best-effort/excess buffer may be filled with old excess packets and all of them has to be dropped. Hence dequeuing may take $O(N_{eb})$ time. However, the dequeue time is still $O(1)$ per packet and we amortize the cost of the dequeuing over the enqueueing. Hence the enqueue is charged the dequeue cost and we thus have $O(1)$ amortized time.

One major problem with this solution is that excess packets will almost always be too old when they arrive at the front of the buffer. Since most of the best-effort traffic uses TCP/IP and this tends to fill the buffers to the limit if the load is relatively high, the time for any packet to reach the front of the buffer roughly equals $N_{eb}p/b$ where p is the average packet size and b is the bandwidth on the out link. If this time is greater than the maximum allowed delay for conforming/excess packets, the excess packets will be dropped since they are too late.

2.3 IT implemented using queue tickets

To improve the handling of the problem with the naive solution we extend the solution further introducing *queue tickets* (\mathcal{Q}). Queue tickets are found at, for example post offices. A queue ticket is an order indication but it is not for the arrival order, instead it is for the order in which customers are to be served. Just as at the post office one may give her ticket to another customer, we allow one excess packet to use another excess packet's ticket. We let the queue ticket be an indication of which type of packets should be served and store them in a separate buffer.

This time we choose to use one buffer for each type

of packets and one for the queue tickets. Again the buffers for conforming and excess packets store besides the pointers to the packet also a time stamp and a sequence number. The buffer for the best-effort packets does not need to store a time stamp and a sequence number since it is not important that best-effort packets leave in time and in order with respect to prioritized packets. Hence it only stores the pointer to the packets. We still need to keep track of the sequence number for the last sent conforming packet.

As before, in the naive solution, we have problem with packets arriving too close to each other. We use the same solution as before. This gives a space requirement of $4N_c + 3N_e + N_b + N_q + O(1)$, where N_c , N_e and N_b is the length of the conforming, excess and best-effort buffers respectively and N_q is the maximum number of queue tickets. Since excess packets may be given other's queue tickets it is possible that the ticket buffer stores N_e tickets while the excess buffer is empty. If the best-effort buffer than is filled with N_b packets there will be $N_b + N_e = 2N_{eb}$ queue tickets in the buffer. Hence to be able to store the queue tickets N_q has to be greater than or equal to $2N_{eb}$.

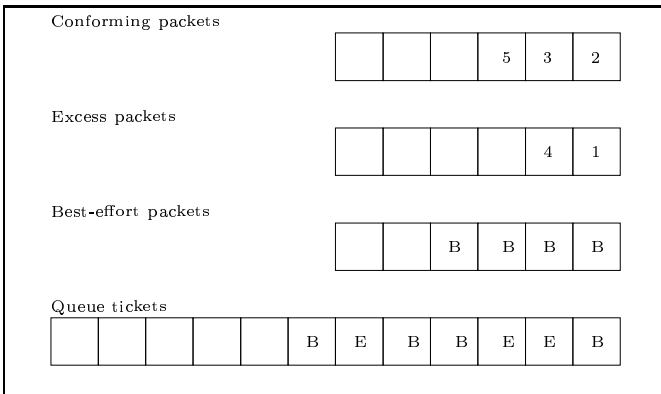


Figure 2: The four buffers storing ordered prioritized (excess and conforming), best-effort packets and queue tickets

Enqueuing of best-effort and excess packets is a little bit more tricky now but still $O(1)$ worst case time. Whenever enqueueing a best-effort or an excess packet we also enqueue its type in the queue ticket buffer. A packet has to be dropped not only if the buffer it should go to is full but also if, for best-effort and excess traffic, a common limit of the number of packets allowed is reached. The limit is used to ensure that best-effort and excess packet are dropped in a fair way when entering the node. This common limit has to be smaller or equal to the length of the smaller of the buffers. This implies that N_e and N_b should be equal and we use N_{eb} to denote their common value. Hence, the space needed by this solution is $4N_c + 4N_{eb} + N_q + O(1)$ compared to $N_c + N_{eb} + O(1)$ and $4N_c + 3N_{eb} + O(1)$ for the EF and naive solutions respectively.

As in the naive solution when deciding from which buffer a packet should be sent, we note that an excess packet should be dropped if it is late $\tau_e < t$ or has a sequence number lower than the last send conforming

packet $\sigma_e < \Sigma$. We have the following steps:

1. Check whether a best-effort or an excess packet is first according too the first ticket Q_q in the non empty queue ticket buffer.
 - (a) In case it is a best-effort packet ($Q_q = best$) check if there is a conforming packet which should be sent before due to time $((t + O) + sendTime(p_b) > (\tau_c + o_c))$. If so dequeue the first packet in the conforming buffer otherwise dequeue the first best-effort packet in the best-effort buffer.
 - (b) In case it is an excess packet ($Q_q = excess$)
 - and the excess buffer in non empty, first check if the first packet in the excess buffer should be dropped.
 - If so drop it but do not drop the queue ticket and start over from 1b.
 - otherwise check if there is a conforming packet which should be sent before (due to sequence number ($\sigma_c < \sigma_e$) or time $((t + O) + sendTime(p_e) > (\tau_c + o_c))$). If so dequeue the conforming packet otherwise dequeue the excess packet.
 - If there is no packet in the excess buffer remove the queue ticket and start over from 1.
2. If the queue ticket buffer is empty neither the excess nor best-effort buffer can be non empty. Hence dequeue the first packet in the conforming buffer unless it is empty in which case we return some predefined value.

Which concludes the description of the *ticket* solution.

3 Functional Tests

To test the functionality of our proposals we have implemented them in NS-2. The network topology (Figure 3) we used has n pairs (P_i, p_i) of nodes sending/receiving prioritized traffic and m pairs (B_j, b_j) of nodes sending/receiving best-effort traffic. All prioritized traffic go through node A_1 which tags the packets as either conforming or excess packets. All best-effort traffic go through node A_2 which tags the packets as best-effort packets. The packets are then forwarded to the *IT* node which implements the IT PHB. Bandwidth and delay of the links can be seen in Figure 3, note that the link after the *IT* node is the bottle-neck.

In our first round of simulations we fixed the load by using one prioritized and one best-effort flow with constant bit-rate sent using UDP and we varied the maximum allowed delay in the IT node. The buffers in the IT node where set to hold at most 32 packets which at an average packet size of 1000 bytes gives an approximate delay of 25 ms if no conforming traffic is sent. The delay will increase as the proportion of conforming traffic increases. All the graphs are an average of ten simulations.

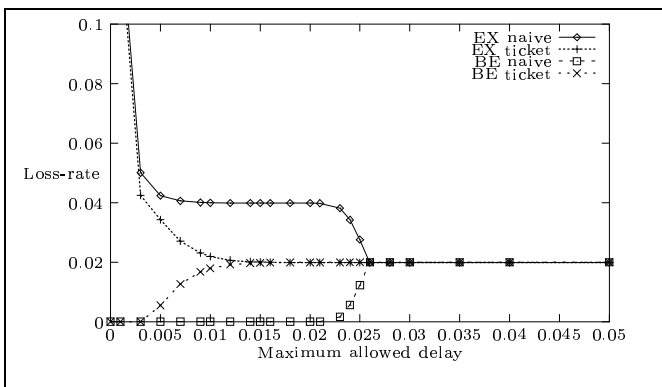
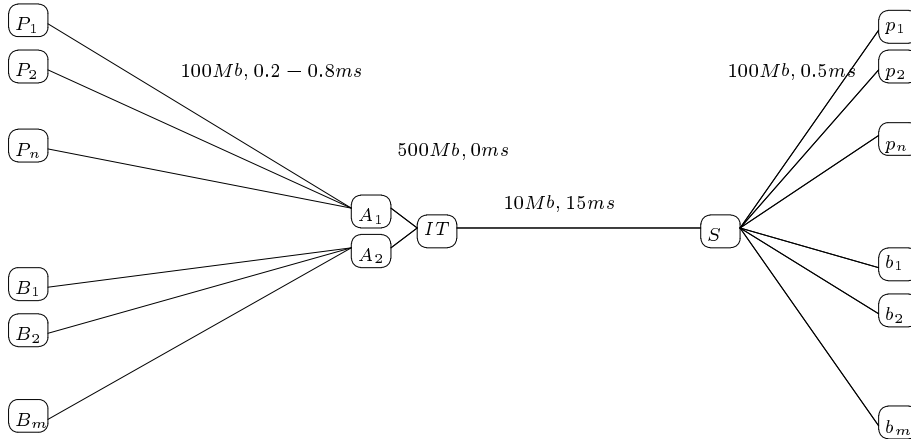


Figure 4: Loss-rate, at 5.1Mb best-effort and 5.1Mb prioritized (0Mb conforming), in the IT node.

Figure 4 shows the loss-rate for best-effort and excess packets when the over load on the link is 2% and none of the prioritized traffic is conforming. Hence the proportion of dropped packets above 2% is dropped since they are late and the proportion below due to over load. We see that the naive solution drops only excess packets when the allowed delay is less than the queue delay (25 ms) as we expected. Moreover, the queue ticket solutions manages to handle this case better.

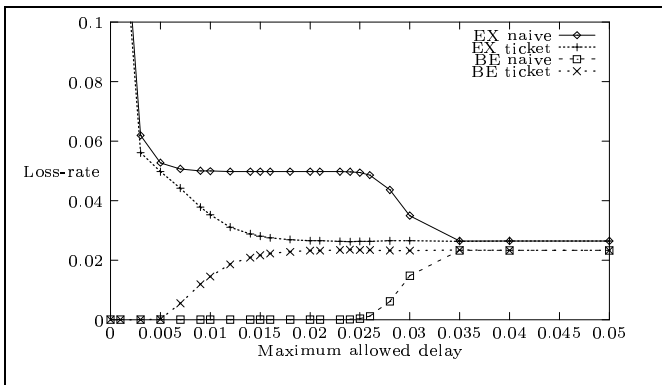


Figure 5: Loss-rate, at 4.1Mb best-effort and 6.1Mb prioritized (2Mb conforming), in the IT node.

Figure 5 show the same situation but with 2Mb of the prioritized traffic as conforming and hence it is not allowed to be dropped. This leaves only 8Mb for excess

and best-effort traffic and hence the proportion of packets that need to be dropped is $0.2/8.2 = 2.5\%$. We see here that neither the naive nor the queue ticket solution manages to achieve equal loss-rate ratio for best-effort and excess packets. This is due to the fact that the bit-rate is greater on the in-link than on the out-link of the IT node. Hence prioritized packets can arrive too close in time and we do not solve the problem for excess packets but drop them. We got similar results with different loads and packet sizes.

In our second round of simulation we used a number of FTP flows on TCP/SACK as best-effort traffic and *TCP-Friendly Rate Control (TFRC)* flows as prioritized traffic. TFRC is proposed as an *Equation-Based Congestion Control* by Floyd et al. [3]. It was developed with real time application, which suffers from big variations in bit-rate, in mind. Since our goal is to provide in-order forwarding of excess traffic in respect to conforming traffic we focus on the case where we have conforming traffic (2Mb out of the available 10Mb).

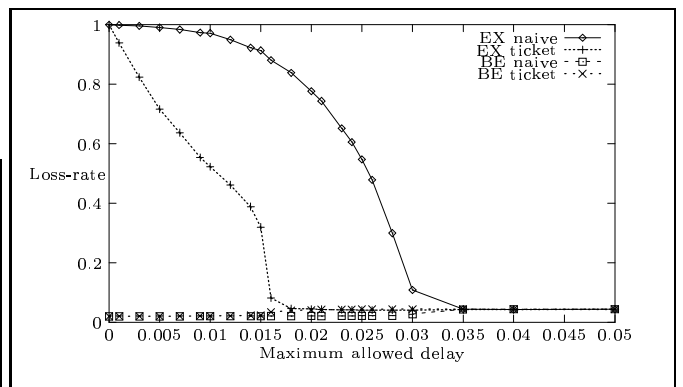


Figure 6: Loss-rate, using 7 best-effort flow and 7 prioritized flows.

Figure 6 show the loss-rate when 7 FTP flows and 7 TFRC flows are used and Figure 7 show the corresponding bit-rate. The bit rate for the excess traffic is computed as the total bit-rate for prioritized traffic subtracted by the amount of conforming traffic. Also the bit-rate is presented as average per flow, i.e. the total amount divided by the number of flows.

We especially note that we have focused on equaliz-

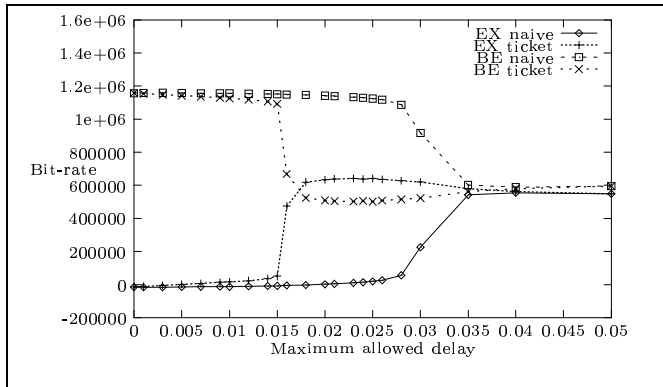


Figure 7: Bit-rate, using 7 best-effort flow and 7 prioritized flows.

ing the loss-rate ratio for best-effort and excess packets. However, the steady state sending rate for TCP and consequently TFRC is based on a function of packet size, loss-rate and round trip time. We provide delay guarantees for the prioritized flows giving shorter round trip time, which means that the prioritized flows get a higher send bit-rate (which we also see in Figure 7).

Furthermore, we have seen that the loss-rate ratio tend to vary with the ratio between best-effort and prioritized traffic. We believe the choice of ratio of TFRC and TCP flows affects the loss-rate since TCP is more burtsy than TFRC [3].

4 Conclusion, Open Questions and Acknowledgment

We have seen that we can provide IT, however we still need to evaluate the queue ticket solution in greater detail. First we need to evaluate the relation between the ratio of best-effort compared to prioritized traffic and the loss-rate ratio. To do this we first need to determine whether the choice of TCP or TFRC flows for prioritized flows is significant or not.

Currently a pure drop tail strategy is used to drop packet when the buffers become congested. We believe that using *random early detection (RED)* [4] to avoid congestion is preferable and it should be evaluated.

Moreover, the ticket queue solution might be improved by using *proportional loss-rate differentiation* [1, 2] to decide if a best-effort or an excess packet should be dropped.

Another improvement might be to keep track of the sequence number for the last sent conforming packet per flow instead of for the aggregate. This should reduce the number of out-of-order drops for the excess packets.

Acknowledgement

Finally we would like to thank Svante Carlsson and Olov Schelén for initial discussions about differentiated delay guarantees and some helpful ideas on how to go ahead.

References

- [1] Ulf Bodin, Andreas Jonsson, and Olov Schelén. On creating proportional loss-rate differentiation: Predictability and performance. To appear in *IWQoS 2001.*, 2001.
- [2] Constantinos Dovrolis and Parameswaran Ramanathan. Proportional differentiated service, part II: Loss rate differentiation and packet dropping. In *Proceedings of 1999 Seventh International Workshop on Quality of Service (IWQoS'99)*, London, UK, June 1999.
- [3] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000 Conference*, volume 30 (4) of *Computer Communications Review*, pages 43–56, Stockholm, Sweden, October 2000. ACM SIGCOMM.
- [4] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.
- [5] Van Jacobson, Kathleen Nichols, and Kedarnath Poduri. An expedited forwarding phb. IETF RFC 2598, June 1999. ftp://ftp.ietf.org/papers/ef_phb.pdf.
- [6] NS (Network Simulator). <http://www.isi.edu/nsnam/ns/>.