

Extended Expedited Forwarding: the In-Time PHB group

Johan Karlsson [†] Ulf Bodin [†] Andrej Brodnik ^{†*}
Andreas Nilsson [†] Olov Schelén [†]

Abstract

This paper presents a new set of forwarding behaviors that fits rate-adaptive and delay-sensitive applications with limited loss tolerance. We consider an application to have limited loss tolerance if it needs loss-free forwarding of specific packets up to a certain rate. The new set of forwarding behaviors are attractive for developing real-time applications for the Internet. In particular, such applications can be designed to use reserved forwarding capacity efficiently and compete for more bandwidth while being fair to best-effort traffic. To provide the new set of forwarding behaviors, we define a scheduling mechanism that can be implemented efficiently. Through simulations, we show that this mechanism supports the defined forwarding behaviors.

1 Introduction

Real-time applications are becoming increasingly common in the Internet, e.g. video and voice over IP. Such applications need to present data to users with short delay (i.e., they are delay-sensitive). Real-time applications may also prefer low loss-rate since error resilience then can be traded off for better compression efficiency and higher quality. Applications optimized for low loss-rate are said to be intolerant.

Clearly, delay-sensitive and intolerant applications gain from guarantees on bounded delay and loss. Such guarantees can be provided with the Expedited Forwarding (EF) per-hop behavior (PHB) [6]. EF is part of the Differentiated Services (DiffServ) architecture [2].

EF assumes that traffic using the PHB is peak-rate limited. This can be achieved by associating users with service profiles. Then, traffic is policed to fit these profiles as it enters DiffServ capable networks. EF traffic *conforming*

*Institut of Mathematics, Physics, and Mechanics Department of Theoretical Computer Science Jadranska 19, 1111 Ljubljana, SLOVENIA

[†]Department of Computer Science and Electrical Engineering, Luleå University of Technology, S-971 87 Luleå, Sweden. E-mail: {k,uffe,brodnik,andreas,olov}@sm.luth.se. Phone +46 920 491000, fax +46 920 492831.

to the peak-rate of each profile is tagged with the for EF selected DiffServ Code Point (DSCP) and can be given loss-free and low delay forwarding, while EF *excess* traffic is dropped (i.e., traffic exceeding this peak rate).

EF fits delay-sensitive and intolerant applications that need a certain rate, but do not gain from more forwarding capacity if available. Delay-sensitive and tolerant applications that needs a minimal rate and can gain from additional capacity are however not well supported by EF. Such applications adapt their sending rate as response to packet loss (i.e., they are congestion-responsive). Still, they may need loss-free forwarding of specific packets up to a certain rate (e.g., to maintain a minimal frame rate). This kind of congestion-responsive and delay-sensitive applications can be said to have limited loss tolerance.

Applications with limited loss-tolerance being congestion-responsive and delay-sensitive include those envisioned for the Datagram Congestion Control Protocol (DCCP) [13, 14] currently considered for standardization by the IETF. These applications are media streaming, Internet telephony, and on-line games. In addition, we expect video conference applications, having these properties, to be developed for DCCP.

We define a new PHB group that fits congestion-responsive and delay-sensitive applications with limited loss tolerance. We name this PHB group In-Time (IT). IT consists of three PHBs; IT-conforming, IT-excess, and IT-background. For IT, we define the following requirements:

- Conforming and excess packets must be forwarded *in-time*. I.e., these packets must not face queuing delays longer than a pre-configured maximum allowed delay, d .
- Conforming and excess packets collectively must be forwarded *in-order*. I.e., in the same order as these packets arrived.
- Excess traffic must be given a loss-rate approximately equal to or higher than the loss-rate given to background traffic. This is the *loss-rate relation* requirement.
- Conforming traffic must be served at the configured rate, or at a rate higher than the configured rate. IT shares this requirement with EF [6]. We refer to it as the *departure rate* requirement.

The *departure rate* requirement enables guarantees on loss-free forwarding for conforming traffic. Such guarantees can be given by limiting the utilization of a network (with known characteristics and a maximum number hops at any path) through traffic conditioners at network edges [1].

As for EF, IT assumes that conforming traffic is peak-rate limited. While EF excess traffic is dropped, IT excess traffic is however tagged and forwarded through traffic conditioners at network edges. Hence, with IT, con-

forming, excess, and background traffic are all forwarded, but with different DSCPs.

As the traditional best-effort service, the IT-background PHB provides no guarantees. IT-background can, but does not need to, be equal to the default PHB [15], which is the best-effort service through a DiffServ compliant node. Then, IT-background and IT-excess shares bandwidth with default traffic, and bandwidth for IT-conforming only needs to be explicitly allocated. We use the term *best-effort* for the IT-background PHB.

Although not being a requirement, an implementation of IT should treat excess and best-effort traffic approximately equal with regard to loss. Thereby, these traffic aggregates can compete for available bandwidth on similar terms. We refer to this as the *loss-rate fairness* demand.

Most real-time applications require data to be ordered before processing. Without ordered forwarding of conforming and excess packets such applications would need to buffer arriving packets to place data in order. Since this introduces delay, we consider the in-order requirement of IT important.

IT contributes with new differentiation properties not supported by any combination of the PHBs currently specified by the IETF. We believe these properties to be attractive in developing real-time applications for the Internet. Without IT, real-time applications need to use reserved capacity only (e.g., provided by EF), or operate without any upper bound on delay. With IT, real-time applications can be designed to both compete fairly with best-effort traffic for more bandwidth and use reserved capacity efficiently.

Existing applications can be extended to benefit from IT. E.g., Bennett et al. presents an error resilient (to packet loss) and scalable compression method for video that is congestion-responsive [17]. The rate is varied using different quantization steps and, when these steps cannot be reduced further, by sub-sampling the input sequence to reduce the frame-rate. With loss-free forwarding up to a certain rate, decreasing the frame-rate can be avoided. In addition, it might be possible to trade off error resilience for better compression efficiency and higher quality for the protected sub-stream.

EF can be supported in output buffered routers with a prioritized queue scheduler (i.e., a strict non-preemptive prioritized queue). Such a scheduler is appealing to EF since it offers low bounds on latency¹. Low latency for conforming traffic is important also with IT. To offer low latency with IT, we extend a prioritized queue scheduler with support for IT. This scheduler, which we name TICKET, offers equal bounds on latency as a prioritized queue scheduler.

Another appealing property of a prioritized queue scheduler is that it can be implemented efficiently. The extensions for IT consume reasonable amounts of memory and have moderate processing overhead on common

¹With a strict non-preemptive prioritized queue the delay bound for conforming traffic is equal to MTU/c , where c is the link speed [4].

platforms (e.g., Intel Pentium III). Hence, with an efficient prioritized queue scheduler as a base, an efficient TICKET scheduler can be implemented.

Through simulations, we show that TICKET meets the requirements of IT. TICKET has one configurable parameter only, i.e. the maximum allowed delay, d . Longer d gives lower loss-rates for excess traffic². A scheduler implementing IT must support very low d to meet delay requirements of real-time applications. Moreover, while supporting such a low d , loss-rates for excess and best-effort traffic must be similar to enable sharing of bandwidth (i.e., the loss-rate fairness demand). The simulations show that with d set to only a few ms, TICKET gives excess and best-effort traffic similar loss-rates.

2 Algorithms and Data Structures

TICKET is created through extensions to a prioritized queue scheduler using two FIFO buffers. We add a third DSCP for excess traffic (i.e., in addition to the conforming and the best-effort DSCPs), and time-stamps and sequence numbers for excess and conforming packets. These time-stamps and sequence numbers are local within each node. With these additions we define a *naive* scheduler that meets the four requirements of IT.

The naive scheduler (with two buffers) can give considerably higher loss-rate to excess traffic than to best-effort traffic (i.e., the loss-rate fairness demand may not be met). This is because these aggregates are forwarded in the same buffer. Then, excess traffic may get only small amounts of bandwidth, which limits the value of IT.

To achieve similar loss-rates for excess and best-effort traffic, we extend the naive scheduler into the TICKET scheduler by adding *queue tickets* and two additional buffers. TICKET meets the loss-rate fairness demand.

A prioritized queue scheduler using two FIFO buffers is described in Sect. 2.1. In Sect. 2.2 we define extensions to this scheduler creating the naive scheduler, and in Sect. 2.3 we define extensions that completes the TICKET scheduler. Then, in Sect. 2.4, we discuss important implementation and design details of TICKET. Finally, in Sect. 2.5, we analyze time and space requirements of TICKET. Commented pseudo codes for the naive and TICKET schedulers respectively are given in Appendix A.

2.1 A prioritized queue scheduler

A prioritized queue scheduler is an appealing implementation of EF since it offers low bounds on latency [4]. With such a scheduler EF packets are forwarded before any best-effort packet. There are several approaches to create

²Loss-rates for excess traffic is however, as required by IT, always equal or larger than for best-effort traffic.

this behavior. We use two FIFO buffers (i.e., one buffer, **c**, for EF packets and another buffer, **b**, for best-effort packets) as base for our TICKET scheduler. This scheduler does not preempt packets, but buffer **c** has absolute priority over buffer **b**. I.e., when one or more EF packets are queued the first one is dequeued as soon as the previous (EF or best-effort) packet is completely sent. If no EF packet is queued, the first best-effort packet is dequeued.

With two FIFO buffers whereof one is prioritized, both enqueueing and dequeuing can be done with the same time complexity as for a single FIFO buffer, i.e., $O(1)$. The space used by this dual FIFO scheduler (in addition to the space used to store packets) is the space needed in the buffers to store pointers to the packets (i.e., $N_c + N_b$ machine words).

2.2 The naive scheduler

Any scheduler supporting IT has to meet its four requirements. The naive scheduler meets the departure rate requirement since it is based on a prioritized queue scheduler (i.e., conforming packets are forwarded through buffer **c** as with the pure prioritized queue scheduler defined in the previous section).

To meet the in-time requirement of IT, we store, with each conforming and excess packet, a time stamp $\tau = t + d$, where t is the current time and d is the maximum allowed delay. From τ , it can be determined when conforming packets must be sent to meet d , and when excess packets are late and should be dropped.

As many excess and best-effort packets as possible should be sent before sending conforming packets. This is to avoid dropping excess packets due to latency and out-of-ordering. Hence, conforming packets can be delayed until they must be sent (i.e., to meet d). Note that conforming packets are only delayed if the outgoing link has an overload.

To meet the in-order requirement of IT, we store, with each conforming and excess packet, a sequence number, σ . With σ , we keep track of the arrival order among these packets. In addition, we store σ of the last dequeued conforming packet, Σ . Excess packets with σ less than Σ are dropped. When σ of the excess packet at the head of buffer **b** is higher than Σ , but less than σ of the conforming packet at the head of buffer **c**, this conforming packet is sent immediately.

When buffer **b** is full, arriving best-effort and excess packets are dropped independent of their DSCPs. Hence, these packets are equally likely to be dropped due to buffer overflow (assuming equal arrival processes). In addition to buffer overflow, excess packets can be dropped due to latency and out-of-ordering. Consequently, excess traffic may experience higher loss-rates than best-effort traffic although the losses caused by buffer overflow hits them equally often. Hence, the naive scheduler meets the loss-rate

relation requirement of IT.

2.3 The TICKET scheduler

Although the naive scheduler meets the requirements of IT, it does not meet the loss-rate fairness demand. The extensions defined in this section provide support for this demand.

As noted above, excess packets are dropped due to buffer overflow, latency, and out-of-ordering. Best-effort packets are, however, dropped due to buffer overflow only. Consequently, excess packets may experience higher loss-rate than best-effort packets. To achieve similar loss-rates for these packet types, the number of drops due to latency and out-of-ordering must be reduced.

Excess packets already being late or out-of-order must be dropped. Thus, to reduce the number of excess packets dropped for these reasons other excess packets must be given precedence following such drop events. TICKET achieves this by letting excess packets jump ahead in the queue when excess packets are dropped due to latency or out-of-ordering.

The jump ahead for excess packets is achieved using separate buffers for excess and best-effort traffic (i.e., buffer \mathbf{b} for best-effort packets, and buffer \mathbf{e} for excess packets). Queue tickets are used to control the order in which buffers \mathbf{b} and \mathbf{e} are served.

Queue tickets are sequence numbers, \mathcal{Q} , for *accepted* excess and best-effort packets³. Hence, buffers \mathbf{b} and \mathbf{e} are scheduled in the same order as excess and best-effort packets are accepted by the TICKET scheduler. Excess packets can however use queue tickets of previously dropped such packets, while best-effort packets use their own tickets only.

Note that since excess packets may use queue tickets of previously dropped excess packets they will jump ahead of best-effort packets that are currently in the buffer. This does not, however, lead to an increased forwarding capacity for excess traffic compared to best-effort since an excess packet that should have been forwarded was dropped.

To ensure that excess and best-effort packets still are dropped fairly at buffer overflow, a common limit, N_{eb} , of the total number of excess and best-effort packets is used. The limit has to be smaller or equal to the length of the smaller of the buffers ($N_{eb} = \min(N_e, N_b)$). Consequently, N_e and N_b should be equal ($N_{eb} = N_e = N_b$).

To summarize TICKET, for each accepted packet we store a sequence number, a deadline (for conforming and excess packets only), an added time offset (for conforming packets only), and a queue ticket (for excess and best-effort packets only).

³Note that tickets are local for each outgoing interface.

When the deadline for the next conforming packet is reached, it is immediately sent. Otherwise, dequeuing is done according to sequence numbers and queue tickets. Note that an excess packet is dropped when it is late ($\tau_e < t$), or has a sequence number lower than the last sent conforming packet ($\sigma_e < \Sigma$). If both the excess and best-effort buffers are empty, the first packet from the conforming buffer is returned. If only one of these buffers is empty, a packet is returned from the non-empty buffer.

2.4 Implementation and design details

2.4.1 Overlapping conforming packets

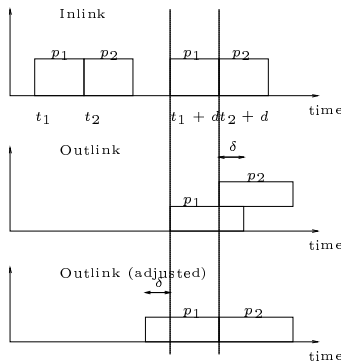


Figure 1: If p_1 is delayed until $t_1 + d$, then p_2 can not be sent before $t_2 + d$. Hence p_1 must be sent at $t_1 + d - \delta$.

Although IT traffic is properly policed, two consecutive conforming packets may arrive to an outgoing link at a rate higher than the out-link rate (e.g., due to packets arriving at different in-links). Then, these packets cannot both be sent in-time if the first one is delayed as long as allowed by d (Fig. 1). To determine for how long, possibly overlapping, conforming packets can be delayed, τ of previous packets can be adjusted for $\delta = \max(((t_1 + d) + \text{sendTime}(p_1)) - (t_2 + d), 0)$.

Since τ of all conforming packets in queue might need to be adjusted, this operation can be expensive. To avoid the cost of making these adjustments, we maintain a time-offset, Δ , for the clock. This offset is added to the current time when deciding which buffer to serve. When a conforming packet arrives that overlap with a packet already in the queue δ is added to the offset Δ . The added time, δ , is stored with the arriving conforming packet so that Δ can be decreased when the packet are dequeued. Since each arriving packet is handled separately, several overlapping packets can be handled in the same way.

2.4.2 Ticket storage

Tickets for best-effort packets can be stored together with these packets. Tickets for excess packets are however preferably stored in a separate buffer, \mathbf{q} . Thereby, tickets for excess packets can remain for future use when a packet is dropped, and even if no excess packet is queued.

Since there can be (at most) one ticket per excess packet, buffer \mathbf{q} should be of equal size as buffer \mathbf{e} (i.e., $N_q = N_{eb}$). Buffer \mathbf{q} can be full although buffer \mathbf{e} has space available to queue packets. Then, however, no ticket has to be allocated for an arriving excess packet since there are already enough tickets in buffer \mathbf{q} .

2.4.3 Loss-rate differences

Loss-rates caused by buffer overflow may differ between best-effort and excess traffic if they have different arrival processes. E.g., the well known bias of drop-tail buffers against bursty flows [11] may cause the loss-rate relation requirement of IT to be violated. Such bias can however be reduced through active queue management [3].

2.4.4 Per-flow out-of-order detection

An excess packet may be in-order within its applications data flow while being out-of-order within the IT aggregate. Hence, using the sequence number for the last sent conforming packet within the same application data flow instead of for the aggregate might reduce the number of excess packet drops caused by out-of-ordering. The sequence number can be stored in a dictionary⁴ with a flow identifier as key.

A per-flow out-of-order detection variant of TICKET was considered in preliminary simulations. These simulations gave the same results as with the pure TICKET scheduler. The per-flow variant may however prove valuable in a scenario with more flows using IT. We consider this as for further studies.

2.5 Time and Space Requirements

The space used by the TICKET scheduler is $4N_c + 3N_e + 2N_b + N_q = 4N_c + 3N_{eb} + 2N_{eb} + N_{eb} = 4N_c + 6N_{eb}$ (i.e., in addition to the space used to store packets). This is merely six words per packet, which is a minor overhead since packets often consists of several hundred of words (an integer word is equal to four bytes on most architectures).

The time complexity for enqueueing/dequeueing is $O(1)$. Both enqueueing and dequeueing consists of, in addition to FIFO buffer manipulations, a

⁴A dictionary can be implemented using, for example, *perfect hashing* [7, 12], where operations can be performed in $O(1)$ amortized time.

computation of the sending time for a packet. This computation involves a division, which however can be avoided with a pre-computed table.

The calculation of the sending time in dequeue can be avoided by allowing an additional delay of conforming packets. In enqueue the sending time is used to separate packets that arrive so close in time that the deadline cannot be met for the second packet. The sending time is however not needed if conforming packets always are separated with the sending time of a MTU sized packet (i.e., all conforming packets are assumed to be of MTU size). This may however have a negative influence on the excess traffic since conforming packets will be sent earlier. Moreover, no delay guarantees can be given for time scales shorter the sending time of an MTU sized packet. We consider this as for further study.

The current time has to be provided by the hardware. However, since this time is only used to ensure that a prioritized packet is not delayed more than allowed, it does not necessarily has to be provided by a real-time clock. For example, the number of hardware clock cycles elapsed since boot time is sufficient (given that the speed of the hardware is known in advance). Thereby, the current time is accessible simply through a register read.

3 Evaluation

In this section, we present simulations evaluating the TICKET scheduler. The simulations are made with the network simulator version 2 (NS-2) [16].

3.1 Simulation Setup

The topology (Fig. 2) supports RTTs between 12 ms and approximately 250 ms (including queuing delay), which is a common range in the Internet [9]. Queuing delays up to 256 packets occur at link CR(a)-CR(b) where the TICKET scheduler is used, and up to 128 packets at link CR(b)-CR(a) where a FIFO scheduler is used. Link CR(b)-CR(a) is assigned less buffers to make queuing delays of the two congested links similar (the average packet size is larger at that link).

Access link rates and delays are reconfigured randomly with values between 22 and 32 Mbps, and 0.1 and 0.9 ms respectively. Feldmann et al. used similar values to emulate switched Ethernet [8]. A positive consequence of making these reconfigurations is that they reduce the risk of flows being synchronized.

Hosts h(07) through h(15) have one TCP Friendly Rate Control (TFRC) [10] connection with each of the hosts h(01) through h(03) (i.e., 27 TFRC connections totally)⁵. The traffic sources at h(01) through h(03) have un-

⁵TFRC can be negotiated as the congestion control mechanism for a DCCP [14] connection.

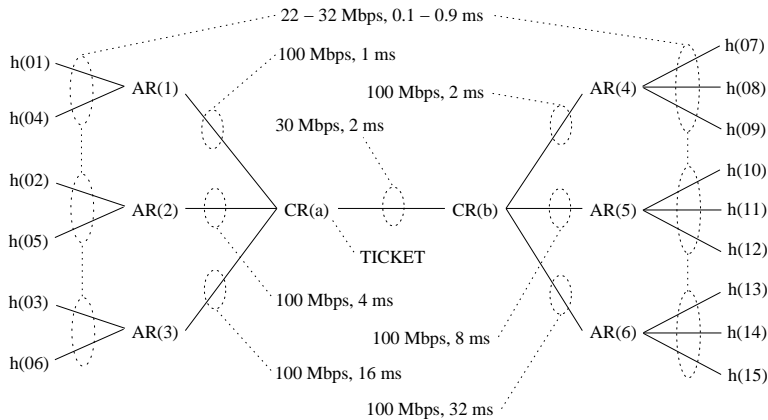


Figure 2: Simulated topology.

limited amounts of data to send. Since IT aims at supporting real-time applications, we set the packet size to 320 bytes⁶.

At AR(1) through AR(3), service profiles of 2.4 Mbps (i.e., 267 kbps/flow) are used to police TFRC traffic into conforming and excess traffic with Time Sliding Window (TSW) [5] based traffic conditioners.

270 Pareto distributed ON-OFF sources at h(04) through h(6) are used to overload link CR(a)-CR(b). From these hosts, h(07) through h(15) download data using TCP Sack and the best-effort service. Packet sizes are up to 1500 bytes. Three levels of overload is created with different average *OFF* period lengths. These overloads cause loss-rates at link CR(a)-CR(b) of 0.7, 2.4 and 5.1 percent in average when measured over 180 s.

In the Internet, TCP ACK packets are likely to be forwarded together with larger data packets. This can influence the spacing between ACKs and thus the burstiness of TCP sources. Also, lost ACKs make TCP sources burstier. To simulate varying ACK spacing and lost ACKs, link CR(b)-CR(a) is overloaded with 270 Pareto distributed ON-OFF sources at h(07) through h(15). From these hosts, h(04) through h(06) download data using TCP Sack. The loss-rates at link CR(b)-CR(a) are 0.9, 1.5, and 1.9 percent in average when measured over 180 s. ACKs from h(04) through h(06) are forwarded together with data traffic over link CR(a)-CR(b), which reduces the average packet size at that link to approximately 650 bytes.

Loss-rates, queuing delays, and throughputs are measured between 20 and 200 s. Results on loss-rates and queuing delays are calculated with 95 percent confidence intervals. These intervals are vary tight for all graphs but the one in Fig. 6. Therefore, we give confidence intervals for this graph only. For each overload, 20 different maximum allowed delays, d in the span between 0 and 100 ms are evaluated.

⁶PCM coded data (i.e., 64 kbps) gives a payload size of 320 bytes with 40 ms packets.

3.2 Modest overload

At a loss-rate of 0.7 percent in average, the TFRC flows get up to 390 kbps of excess throughput in average⁷ at d higher than 40 ms. Hence, with IT, these flows get a considerable higher throughput compared to if they would have used EF with the same amount of allocated forwarding capacity (i.e., 267 kbps).

The additional throughput for the TFRC flows comes at the price of higher delays for both conforming and excess traffic (assuming that EF can give close to zero queuing delay to conforming traffic). The excess throughput per flow is however 360 kbps already at a d of 2 ms. Moreover, average delays at d less than 50 ms are considerable lower than the maximum delays (which are equal to d). This indicates that delays wont accumulate over a path to the sum of all d at overloaded links⁸.

Maximal delays of IT traffic are equal to d up to 50 ms (Fig. 3). Average delays are however considerable less than this maximum. Delays experienced by best-effort traffic decreases as d increases up to 50 ms. This is because the average size of packets in queue decreases (i.e., the fraction of smaller excess packets increase with d).

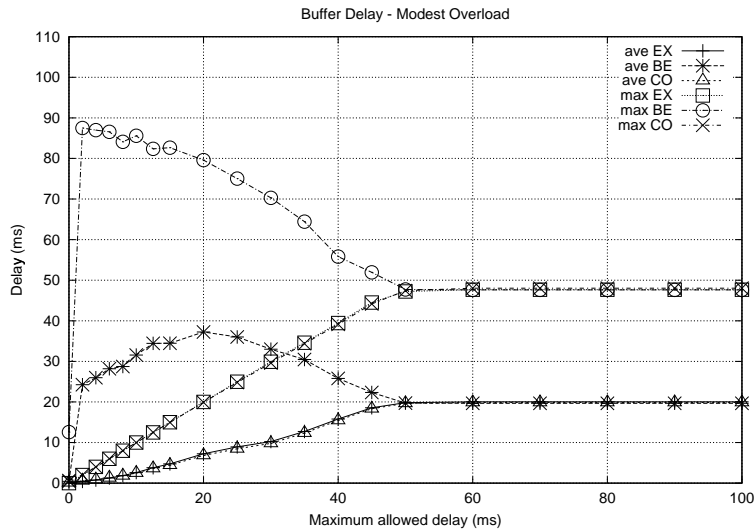


Figure 3: Modest overload.: Queuing delay.

As required by IT, excess traffic is given loss-rates approximately equal to or higher than the loss-rates given to best-effort traffic (Fig. 4). At low d excess packets occasionally need to be dropped due to latency or out-

⁷Graphs on throughput are not shown due to limited space.

⁸As mentioned in Sect. 2.2, conforming and excess packets are delayed at overloaded links only.

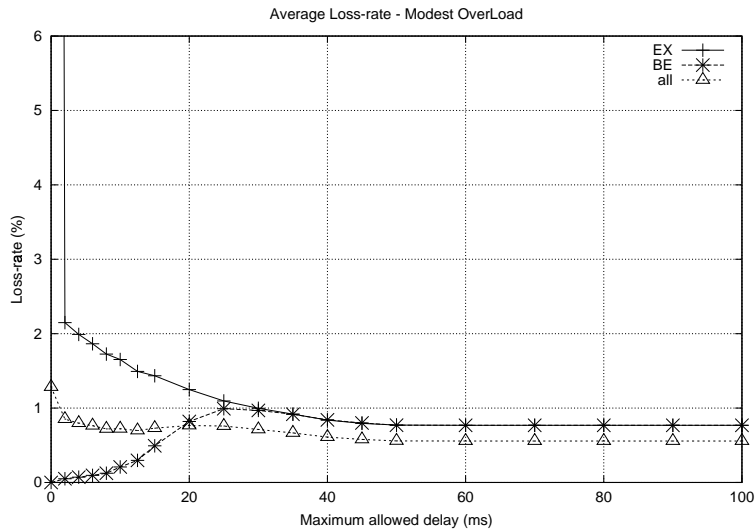


Figure 4: Modest overload.: Loss-rate.

of-ordering. Hence, loss-rates are higher for excess traffic at low d . The loss-rate is however acceptable even for d as short as 2 ms (i.e., 2.2 percent).

3.3 Moderate overload

At a loss-rate of 2.4 percent in average, each TFRC flow gets approximately 110 kbps of excess throughput for all d higher than 2 ms. Hence, larger fractions of the queued packets are best-effort packets, which in average are larger than excess packets. Together with the higher load, this causes generally higher delays than for the load evaluated in the previous section (Fig. 5).

The average delays for excess and conforming traffic are close to zero for d up to 35 ms (Fig. 5). At low d most forwarded excess packets uses queue tickets of previously dropped excess packets. Hence, at low d , excess packets are forwarded almost immediately (or dropped). Conforming packets are also forwarded early to stay in order with excess packets.

Loss-rates are high at d lower than 35 ms. At these d , the fractions of large best-effort packets in queue are high (i.e., few excess packets are queued since many such packets are immediately forwarded). This cause higher delays than when a more even mix of excess and best-effort packets is queued (Fig. 5⁹). High delays means less queue space to absorb bursts. Consequently, more packets are dropped when traffic is bursty.

⁹Only best-effort packet see this delay since many excess packets are forwarded immediately.

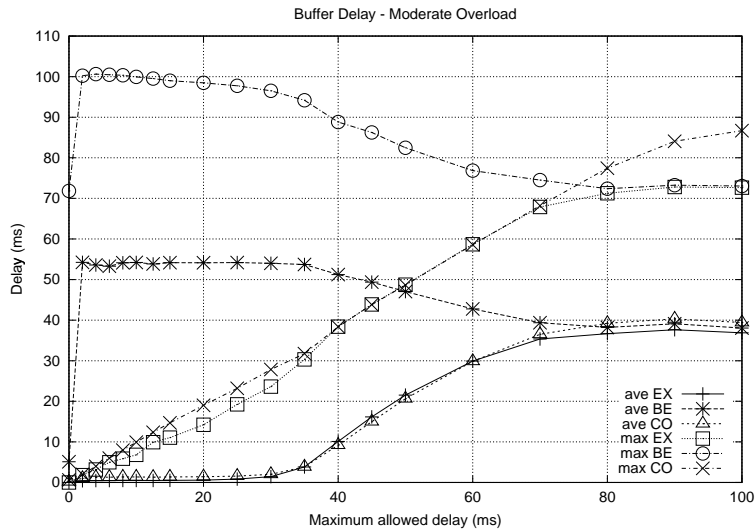


Figure 5: Moderate overload.: Queuing delay.

We expect an additional reason for high loss-rates at low d to be low delays for TFRC flows. At low delays, these sources need higher loss-rates to get similar rates as with higher delays¹⁰. This assumption is supported by the fact that the average throughput of TFRC flows is similar for all d between 2 and 100 ms.

3.4 Severe overload

At a loss-rate of 5.1 percent in average, each TFRC flow gets approximately 40 kbps of excess throughput for all d higher than 2 ms. Delays and loss-rates are slightly higher than at moderate overload, but follows the same patterns.

At d less than 40 ms, best-effort traffic experiences higher loss-rates than excess traffic. The 95 percent confidence intervals do however overlap. The higher loss-rates for best-effort traffic can be explained by the bias against bursty flows of the drop-tail strategy [11]¹¹. As mentioned in Sect. 2.3, TICKET only offer equal ratios of accepted arrivals and successful transfers. This means that differences in burstiness can cause the loss-rate relation requirement of IT to be violated.

¹⁰As with TCP, rates of TFRC sources increase as loss-rates and RTTs decreases.

¹¹TCP is more bursty than TFRC [10].

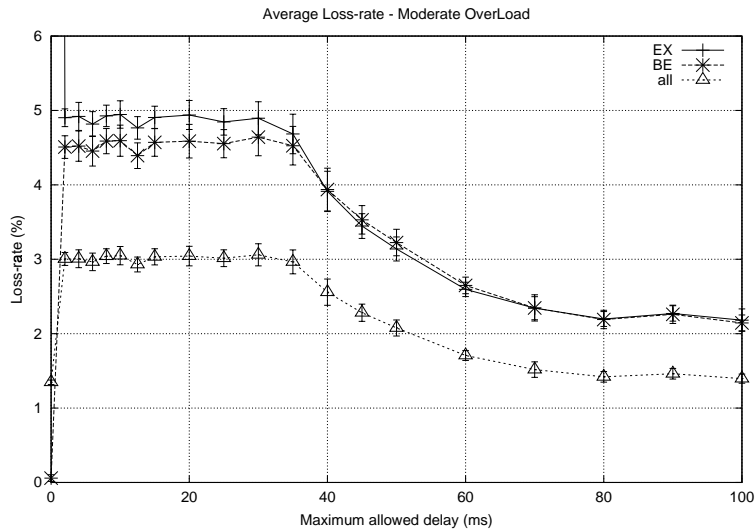


Figure 6: Moderate overload.: Loss-rate.

3.5 Summary of the evaluation

The simulations show that IT can be implemented with the TICKET scheduler. With TICKET, excess traffic get a useful amount of bandwidth since it is given loss-rates close to the loss-rates best-effort traffic experiences. The simulations indicate that excess and best-effort traffic can be given similar loss-rates at d as short as 2 ms at a 30 Mbps link.

At high load, low d cause higher loss-rates to excess and best-effort traffic than with high d . Reasons for these higher loss-rates are that more data is being queued and TFRC sources generates higher load at low d than at high d .

The simulations indicate that excess traffic may be given lower loss-rates than best-effort traffic. The differences are however small and we therefore do not consider them to violate the loss-rate relation requirement of IT.

4 Conclusion

This paper presents the In-Time (IT) PHB group. IT is justified by needs for extensions of EF enabling delay limited forwarding of excess packets in-order with conforming packets.

In addition to the departure rate requirement of EF, IT requires delay limited and in-order forwarding of conforming and excess packets. Also, the loss-rate of excess traffic is required to be approximately equal to or higher than the loss-rate of background traffic (e.g., best-effort traffic). Finally, to enable fair sharing of bandwidth between excess and background traffic,

an implementation of IT should treat these aggregates approximately equal with regard to loss.

We present the TICKET scheduler that implements IT. This scheduler consumes reasonable amounts of memory and have moderate processing overhead on common platforms (e.g., Intel Pentium III). Through simulations we show that TICKET meets the requirements of IT. Moreover, we show that it gives excess and background traffic similar loss-rates at low delay limits.

References

- [1] J. Bennett, K. Benson, A. Charny, W. Courtney, and J. L. Boudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, US, 22–26 Apr. 2001. <http://www.ieee-infocom.org/2001/paper/101.pdf>.
- [2] S. Blake, D. Black, M. Carlsson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475, Dec. 1998.
- [3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. IETF RFC 2309, Apr. 1998.
- [4] A. Charny, J. Bennet, K. Benson, J. Boudec, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Kalmanek, and K. Ramakrishnan. Supplemental information for the new definition of the ef phb (expedited forwarding per-hop behavior). IETF RFC 3247, Mar. 2002.
- [5] D. D. Clark and W. Feng. Explicit allocation of best-effort traffic. *IEEE/ACM Transaction on Networking*, 6(4):362–373, Aug. 1998.
- [6] B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding phb (per-hop behavior). IETF RFC 3246, Mar. 2002.
- [7] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–791, 1994.
- [8] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '98 Conference*, volume 29 (4) of *Computer Communications Review*, pages 301–313, Boston, Massachusetts, US, Aug. 1999. ACM SIGCOMM.
- [9] S. Floyd. Questions. <http://www.aciri.org/floyd/questions.html>, June 29 2001.
- [10] S. Floyd, M. Handley, J. Padhye, and J. 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, Oct. 2000. ACM SIGCOMM.
- [11] S. Floyd and V. Jacobson. Traffic phase effects in packet-switched gateways. *Journal of Internetworking:Practice and Experience*, 3(3):115–156, Sept. 1992.

- [12] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, July 1984.
- [13] E. Kohler, M. Handley, and S. Floyd. Problem statement for dccp. IETF INTERNET-DRAFT draft-ietf-dccp-problem-00.txt, Oct. 2002. <http://www.ietf.org/internet-drafts/draft-ietf-dccp-problem-00.txt>.
- [14] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Datagram congestion control protocol (dccp). IETF INTERNET-DRAFT draft-ietf-dccp-spec-00.txt, Oct. 2002. <http://www.ietf.org/internet-drafts/draft-ietf-dccp-spec-00.txt>.
- [15] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated service field (ds field) in the IPv4 and IPv6 headers. IETF RFC 2474, Dec. 1998.
- [16] NS (Network Simulator). <http://www.isi.edu/nsnam/ns/>.
- [17] W. tian Tan and A. Zakhor. Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE Transactions on Multimedia*, 1(2):172–186, 1999.

Appendix

A Algorithm

We will now describe the two dequeue algorithms (naive and TICKET) in detail. First, a few notations, n_f is the number of elements in the buffer f (c , e and b for the confirming, excess and best-effort buffer respectively and b for the buffer common to excess and best-effort packets in the naive scheduler). For other variables we use subscript f to indicate that the variable refers to a value for the first packet of respective buffer while p_f refers to the first packet it self, e.g., τ_c is the deadline for p_c and τ_e is the deadline for p_e . Δ is the time-offset which the clock should be adjusted for, while δ_c is the amount Δ was increased by for p_c . σ_f is the sequence number of the first packet in buffer f while Σ is the sequence number of the last sent conforming packet. Finally, $f.\text{dequeue}()$ removes and returns the first item from buffer f .

A.1 The naive scheduler

Whenever dequeuing (Algorithm 1) a conforming packet (e.g. line 10) we also need to save the sequence number Σ and adjust Δ (line 8,9). Now first of all, we return a special value (0 or NULL) if all buffers are empty (line 6). If there are no excess or best-effort packets in the buffers we return the first conforming packet (line 10). At line 13 we know there is at least one packet in the excess–best-effort buffer (see line 4).

Now it might be the case that we have to dequeue a conforming packet instead of the excess packet (due to order (line 17) or deadline (line 18) issues. Further, if the next packet is an excess packet we first have to make sure that it should not be dropped (out-of-order or late) (line 14). If not we dequeue and return the excess packet (line 23).

If the next packet to be dequeue is a best-effort packet, it might, again, be the case that we have to dequeue a conforming packet instead (due to deadline (line 27)). Otherwise we just dequeue and return the best-effort packet (line 32).

If we dropped an excess packet (line 15) no packet has been returned and we start over from the top (line 35). Note that we only start over if a packet has been dropped, hence for each loop we remove 1 packet from the buffers. In each loop only a constant amount of work is done. Hence dequeuing is done in $O(1)$ time per removed packet.

A.2 The TICKET scheduler

In the TICKET scheduler (Algorithm 2) we also use queue tickets and we let Q_f denote the queue ticket for the first packet in buffer f . To simplify the code we let $Q_f = \infty$ if $n_f == 0$. Remember that the queue tickets for excess packets are stored in a separate buffer and not with the packet it self.

The main difference from the naive scheduler is that excess and best-effort packets are not stored in the same buffer and we use queue tickets to decide which kind to send (line 13).

A queue ticket is only removed when it has been used (line 23). Hence even if an excess packet is dropped (line 15) the queue ticket remains and may be used by the next excess packet.

```

Packet *dequeue()                                     /* 1 */
BEGIN
  DO
    IF ( $n_b == 0$ ) THEN
      IF ( $n_c == 0$ ) THEN
        return 0;
      ELSE
         $\Sigma = \sigma_c$ ;
         $\Delta = \Delta - \delta_c$ ;
        return  $c.dequeue().p$ ;                               /* 10 */
      END
    END
    IF (type( $p_b$ ) == excess) THEN
      IF ( $n_c > 0$  AND
          ( $\tau_c < ((t + \Delta) + sendTime(p_b))$  OR
            $\sigma_c < \sigma_b$ ))
         $\Sigma = \sigma_c$ ;
         $\Delta = \Delta - \delta_c$ ;
        return  $c.dequeue().p$ ;
      ELSIF ( $\sigma_b < \Sigma$  OR  $\tau_b < t$ )                    /* 20 */
        drop( $p_b$ );
      ELSE
        return  $b.dequeue().p$ ;
      END
    ELSE
      IF ( $n_c > 0$  AND
           $\tau_c < ((t + \Delta) + sendTime(p_b))$ )
         $\Sigma = \sigma_c$ ;
         $\Delta = \Delta - \delta_c$ ;
        return  $c.dequeue().p$ ;                               /* 30 */
      ELSE
        return  $b.dequeue().p$ ;
      END
    END
  WHILE (TRUE);
END

```

Algorithm 1: Dequeuing in an IT naive node.

```

Packet *dequeue()                                     /* 1 */
BEGIN
DO
  IF ( $n_e == 0$  AND  $n_b == 0$ ) THEN
    IF ( $n_c == 0$ ) THEN
      return 0;
    ELSE
       $\Sigma = \sigma_c$ ;
       $\Delta = \Delta - \delta_c$ ;
      return  $c.dequeue().p$ ;                               /* 10 */
    END
  END
  IF ( $Q_e < Q_b$ ) THEN
    IF ( $n_c > 0$  AND
        ( $\tau_c < ((t + \Delta) + sendTime(p_e))$  OR
          $\sigma_c < \sigma_e$ ))
       $\Sigma = \sigma_c$ ;
       $\Delta = \Delta - \delta_c$ ;
      return  $c.dequeue().p$ ;
    ELSIF ( $\sigma_e < \Sigma$  OR  $\tau_e < t$ )                 /* 20 */
      drop( $p_b$ );
    ELSE
       $q.dequeue()$ ;
      return  $e.dequeue().p$ ;
    END
  ELSE
    IF ( $n_c > 0$  AND
         $\tau_c < ((t + \Delta) + sendTime(p_b))$ )
       $\Sigma = \sigma_c$ ;
       $\Delta = \Delta - \delta_c$ ;                               /* 30 */
      return  $c.dequeue().p$ ;
    ELSE
      return  $b.dequeue().p$ ;
    END
  END
END
WHILE (TRUE);
END

```

Algorithm 2: Dequeuing in an IT TICKET node.
