Drop Strategies and Loss-rate Differentiation

Ulf Bodin and Olov Schelén
Luleå University of Technology, SE - 971 87 Luleå, Sweden,
{uffe, olov}@sm.luth.se

Abstract

When offering loss-rate differentiation in IP networks, the drop strategy used can have a considerable influence on packet loss and delay. In particular, a strategy of dropping packets only as they arrive can cause bursty loss patterns and high jitter. When only arriving packets are dropped, the router may need to wait for low priority packets to arrive before dropping any packet. This results in larger queue oscillation than if low priority packets were dropped immediately from the queue. Queue oscillation gives bursty loss patterns and delay jitter. We present simulations showing that dropping packets from the queue gives smoother loss patterns and less jitter than if packets are dropped only as they arrive. These simulations cover both TCP Sack and TCP Reno. WRED with and without the gentle modification is used to make drop decisions.

1. Introduction

The Internet Engineering Task Force (IETF) has defined architectural extensions to support service differentiation on the Internet. The Differentiated Services (DiffServ) architecture [1][2] includes router mechanisms for differentiated forwarding.

With DiffServ, drop precedence levels can be assigned to IP packets. Differentiation between such levels is part of the Assured Forwarding (AF) per-hop behavior (PHB) group [8]. AF can be used to offer differentiation among congestion-responsive applications (e.g., applications using TCP). The traffic of each user is tagged as being *in* or *out* of their service profiles. Packets tagged as *in*-profile are assigned lower drop precedence than those tagged as *out*-of-profile.

Loss-rate differentiation can be created with queue mechanisms (e.g., RED [5] based queue mechanisms such as Weighted RED (WRED) [4], RED In and Out (RIO) [3] and WRED with Thresholds (WRT) [10]). We show that the drop strategy used by such a mechanism can have a considerable influence on packet loss and delay.

The most common drop strategy used by differentiating queue mechanisms is "*drop-arrivals*" (also known as "*drop-tail*"). With this strategy, packets are dropped only as they arrive (i.e., already queued packets are never dropped).

When offering loss-rate differentiation, the drop-arrivals strategy can cause drops to be delayed. For example, say that the detected level of congestion at a particular link indicates that packets should be dropped to perform early congestion signaling[1]. However, for a period, all arriving packets are tagged with a drop precedence level at which packets should not be dropped. With drop-arrivals, the router has then to wait until packets tagged with higher drop precedence levels arrive and drop them instead.

Delayed drops can result in burstier loss patterns and more jitter compared to dropping packets immediately at congestion. When drops are delayed, TCP reduces its sending rate later than if drops were made immediately when congestion first occurred. Moreover, TCP increases its sending rate until packet loss is detected (exponentially at slow-start and linearly at congestion avoidance). With sources behaving as TCP, delayed drops give larger queue oscillation compared to immediate drops. In addition to jitter, queue oscillation gives bursty loss patterns.

The risk of dropping several packets within a TCP source's window of data increases with bursty loss patterns. Such multiple losses can force TCP Reno into slow-start, which may decrease the utilization of congested links. TCP Sack can better handle multiple losses within one window of data [11]. Moreover, the smoothening of loss patterns provided by WRED can be improved by applying the gentle modification of RED to WRED [12]. However, even with TCP Sack and the gentle modification of WRED, delayed drops give burstier loss patterns and more delay variations (i.e., jitter) than immediate drops.

[1] Early congestion signaling makes congestion-responsive applications (e.g., applications using TCP) reducing their sending rate before congestion get more severe. RED [5] provides such signaling.

1092-1658/01 $17.00 © 2001 IEEE

A drop strategy that can offer loss-rate differentiation with low drop delay is "*drop-from-queue*" (also known as "*pushout*"[2]). The drop-from-queue strategy implies that packets most often are dropped immediately at congestion. For example, say that a packet should be dropped, but that the arriving packet is tagged with a drop precedence level at which packets should not be dropped. With drop-from-queue, a packet at higher level is dropped from the queue immediately instead of later when such packet arrives. This presumes that at least one packet that can be dropped is present in the queue. Otherwise, the queue mechanism has to wait until such packet arrives (as with drop-arrivals).

We show, through simulations, that less queue oscillation is obtained with drop-from-queue than with drop-arrivals. In the simulations, all packets are dropped at a high drop precedence level (i.e., *out*-of-profile packets) before dropping any packet at a lower level (i.e., *in*-profile packets). The amount of *in*-profile traffic is kept well below link capacity. Consequently, loss-rate is zero for *in*-profile traffic. The simulations cover TCP Reno and TCP Sack. WRED is used with and without the gentle modification.

2. Creating Loss-rate Differentiation

In this section, we describe how RED [5] based queue mechanisms can be used to create loss-rate differentiation. Moreover, the two different strop strategies evaluated in this paper are presented and their respective computational overhead is discussed.

2.1. Differentiating Queue Mechanisms

As mentioned in Section 1, loss-rate differentiation can be created with differentiating queue mechanisms such as WRED [4], RIO [3] and WRT [10]. These mechanisms all operate on one or more average queue lengths. For example, WRED drops arriving packets tagged with drop precedence level 1 with a probability between zero and max_p(1) when avg_ql is between min_th(1) and max_th(1) (Figure 1). When avg_ql exceeds max_th(1), all such packets that arrive are dropped. Arriving packets tagged with drop precedence level 0 get dropped with a probability between zero and max_p(0) when avg_ql is between min_th(0) and max_th(0). When avg_ql exceeds max_th(0), all arriving packets are dropped[3].

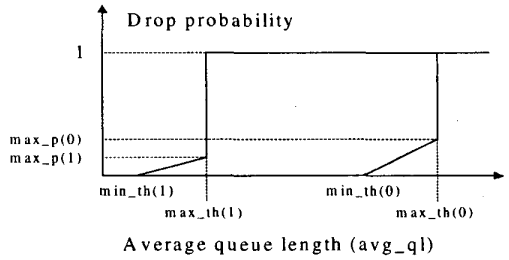

Figure 1 The Weighted RED mechanism.

Dropping packets with a probability that increase with avg_ql is appealing since it provides early congestion signaling. Thereby, congestion responsive sources (e.g., TCP sources) can reduce their sending rates before queues become saturated. Shorter queues mean less delay and more space to absorb bursts. This give lower loss-rates and reduces the risk of locking flows out [6].

The main difference between WRED, RIO and WRT is how they make drop decisions for low drop precedence traffic (i.e., *in*-profile traffic). In contrast to WRED (which uses avg_ql to decide when to drop low drop precedence packets) RIO and WRT calculate a separate average queue length for *in*-packets. RIO uses this variable to make drop decisions for these packets. WRT, on the other hand, uses the separate average queue length to decide when to treat arriving *in*-packets as if they where tagged as being *out*-of-profile. This prevents WRT from starving *out*-of-profile traffic. *Out*-of-profile traffic can get starved with WRED and RIO if *in*-profile traffic is not properly policed. The issue of starvation is discussed in [10]. WRED, RIO and WRT behave equally when only *out*-of-profile packets are dropped.

2.2. Two Different Drop Strategies

The drop strategy used by a differentiating queue mechanism can influence packet loss and delay at congested links. The most common drop strategy used by differentiating queue mechanisms is "*drop-arrivals*" (also known as "*drop-tail*"). This means that packets are dropped only as they arrive (i.e., already queued packets are never dropped).

When offering loss-rate differentiation, dropping packets only as they arrive can cause drops to be delayed. For example, assume that a WRED queue is configured with min_th(1) set to 15 packets, max_th(1) to 45 packets and max_p(1) to 0.1. The major part of the traffic originates from TCP sources. At some point in time, avg_ql is 30 packets. This implies that a drop probability

[2] The pushout drop strategy is extensively evaluated in the context of shared buffer ATM switches (e.g., [14] and [15]). These studies do not however evaluate implications on loss-rate IP packet differentiation for congestion-responsive applications.

[3] WRED provides eight levels of drop precedence, which are numbered from 0 to 7. In this paper, we use level 0 and 1 only.

of 0.05 will be applied to arriving packets at drop precedence level 1. Packets arriving at level 0 will not be dropped at all. Unfortunately, most arriving packets for a period are tagged with level 0 and do not get dropped. Since only a few packets are dropped, many TCP sources continue to increase their sending rate. When more packets at level 1 finally arrive, avg_ql has increased to 50 packets. This means that all arriving packets at level 1 are dropped until avg_ql becomes less than 45 packets. Hence, with few packets arriving at level 1 in times of congestion, the loss-rate can increase dramatically with the drop-arrivals strategy. Moreover, packets at level 1 can be dropped in bursts when avg_ql exceeds max_th(1).

The gentle modification of RED [12] can reduce the problem of high loss-rates with the drop-arrivals strategy. Applied to WRED, this modification can make the drop probability go from max_p(1) to 1 between max_th(1) and twice max_th(1) (Figure 2). This does not however solve the problem of large queue oscillations. Drops can still be delayed, which cause long queues and large queue oscillation. The queue can be even longer with the gentle modification. This is because TCP sources do not reduce their sending rates as much as they would if a burst of packets were dropped when avg_ql exceeds max_th(1).

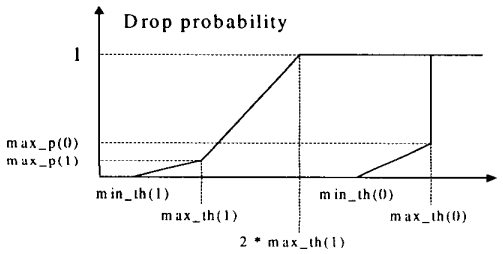

Figure 2 WRED with the gentle modification.

A drop strategy that can offer loss-rate differentiation with lower drop delay is "*drop-from-queue*" (also known as "*pushout*"). Given that there is at least one packet at drop precedence level 1 present in the queue at congestion, such a packet is dropped from the queue when the arriving packet is tagged with level 0. With the same parameter settings as for the previous example, packets at level 1 will be dropped immediately from the queue with probability 0.05 when avg_ql is 30 packets.

The drop-from-queue strategy enables routers to drop packets immediately at congestion. This makes responsive sources such as TCP reduce their sending rate earlier than with the drop-arrivals strategy. Consequently, avg_ql will increase less than with the drop-arrivals strategy. This gives less queue oscillation.

The drop-from-queue strategy can be configured to drop the arriving packet if it is an *out* packet and the last *out* packet present from within in the queue otherwise. Other possible configurations are to drop a random *out* packet or the first *out* packet from the queue. Because of shorter queuing delay, these configurations give less drop delay than dropping the arriving or the last *out* packet. In this paper, we do not examine the strategies of dropping a random *out* packet or the first *out* packet from the queue. This is to minimize effects of shorter queuing delay with drop-from-queue than with drop-arrivals in comparing these drop strategies.

2.3. Computational Overhead

The computational and memory overhead can be expected to be higher for drop-from-queue than for drop-arrivals. The extra overhead occurs when a high precedence packet shall be dropped and a low drop precedence packet arrives to the congested queue. Then, a packet at the high drop precedence level present in the queue has to be found and dropped to make place for the arrived low drop precedence packet. If the first or a random packet at the high drop precedence level shall be dropped from the queue instead of the last one, this extra overhead occurs for all drops (i.e., not only when the arriving packet cannot be dropped). With the drop-arrivals strategy, these operations are not needed at all.

Dropping the last packet at a specific drop precedence level present in the queue is expensive with singe-linked queues since the whole queue may need to be traversed to find the last packet (i.e., if the last packet tagged with the drop precedence level from which a drop shall be made also is the last packet present in the queue). A pointer to the last packet at specific level might make the operation of finding this packet more efficient. The cost of traversing the whole queue must however be compared to the additional cost of maintaining such a pointer. Moreover, traversing the queue can be implemented as a background process. Hence, using a tail pointer might not necessarily make the operation of dropping the last packet at a specific drop precedence level more efficient.

To summarize, it is not obvious how to minimize computational and memory cost of dropping packets from the queue. Moreover, such optimizations depend on router architecture and hardware properties. For example, it has been shown that pushout (i.e., drop-from-queue) can be implemented efficiently in shared memory ATM switches (e.g., [16] and [17]). This may indicate that drop-from-queue can be implemented efficiently in IP routers. In this paper, we consider however analyses of computational and memory overhead in dropping packets from the queue as further work.

3. Simulations

In this section, we present simulations examining loss-rates, link utilization and queue oscillation for WRED with the drop-arrivals and drop-from-queue strategies. The simulations are made with NS [7].

3.1. Simulation Setup

For our simulations, we use a topology for which one link is congested (i.e., link R4 – R3 in Figure 3). At this link, 16 long-lived TCP Reno or Sack connections are used to download FTP data from A and B to receivers r1 through r8. These downloads are started randomly in the first five simulated seconds.

Receiver t1 through t4 download FTP data from C and D. This introduces traffic in the same direction as ACK traffic from receiver r1 through r8. In the Internet, small ACK packets are likely to be forwarded together with larger data packets. Such data traffic can influence the spacing between ACKs so that the burstiness of traffic from A and B is affected[4].

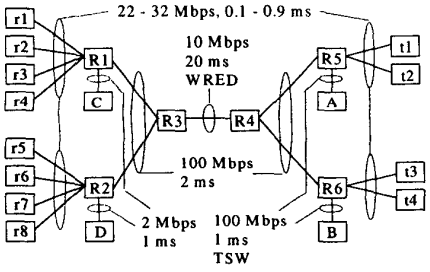

Figure 3 Simulated topology.

WRED is used to differentiate between *in* and *out* packets at link R4 – R3. All other queues are simple FIFO with enough buffer-space to not drop packets (except links C – R1 and D – R2). Traffic tagged as *in*-profile is forwarded at drop precedence level 0 and traffic tagged as *out*-of-profile is forwarded at level 1. WRED is configured with min_th(1) set to 15 packets, max_th(1) to 45 packets and max_p(1) to 0.1. The min_th(0), the max_th(0) and the maximal buffer-space parameters are set large enough to not drop *in*-profile packets.

At A and B, Time Sliding Window (TSW) [3] based traffic conditioners are used to tag packets as *in*-profile up to a certain rate and *out*-of-profile above this rate. For each set of simulations, this rate is varied between 0.0 and 9.2 Mbps in steps of 0.6 Mbps. For each scenario studied, the simulations sets are repeated 32 times with different seeds for the random start of connections. The 32 samples are used to calculate 95 percent confidence intervals for the metrics measured.

In the simulations, loss-rates are measured for all traffic and for *out*-of-profile traffic separately. Effective throughputs are measured for payload data only. Each simulation lasts for 130 simulated seconds. Loss-rate and throughput are measured between 10 and 130 seconds to let the system stabilize before measuring starts.

The bit-rates and delays of links connecting receivers to routers are reconfigured twice every simulated second. Similar values are used in [13] to emulate switched Ethernet. A positive consequence of making these reconfigurations is that synchronization effects among TCP connections get reduced further[5].

3.2. TCP Reno and Standard WRED

For the simulations presented in this section, all sources use the Reno version of TCP. The differentiating queue mechanism is WRED without the gentle modification. Figure 4 shows loss-rates for the drop-arrivals and the drop-from-queue strategy.

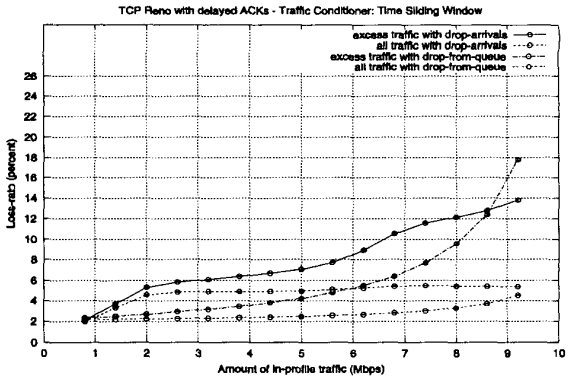

Figure 4 Loss-rates – TCP Reno and St. WRED.

Loss-rates for all traffic is higher for all amounts of *in*-profile traffic simulated with drop-arrivals than with drop-from-queue (Figure 4). However, at low amounts of *in*-profile traffic, these loss-rates are similar with drop-arrivals and with drop-from-queue. This is because most packets arriving are tagged as *out*-of-profile. Hence, with drop-arrivals, the router does not need to wait long for an *out* packet to arrive before being able to drop it.

[4] When there is data to send, the rate at which a TCP source releases segments of data is controlled by the rate at which acknowledgements arrive to the TCP source.

[5] The random drops made by WRED also reduce the risk of having TCP flows synchronize.

Loss-rates for all traffic are similar with drop-arrivals and drop-from-queue also for high amounts of *in*-profile traffic. This is because most packets arriving are tagged as *in*-profile. Consequently, the router does not always find an *out* packet in the queue when the arriving packet is tagged as *in*-profile and an *out* packet should be dropped to signal congestion. The router will then need to wait for an *out* packet to arrive before being able to drop a packet even with the drop-from-queue strategy.

Furthermore, with drop-arrivals, loss-rates of *out*-of-profile traffic increase faster with the amount of *in*-profile traffic below 7 Mbps of *in*-profile traffic than above that amount. This indicates that TCP sources go into slow-start frequently, which reduces the loads generated. TCP Reno enters slow-start when loosing multiple packets within one window of data. At high amounts of *in*-profile traffic, loss-rates of *out*-of-profile traffic can even be higher with drop-from-queue than with drop-arrivals. This indicates that TCP goes into slow-start more seldom with drop-from-queue than with drop-arrivals.

Another consequence of TCP going into slow-start frequently is decreased throughput. Figure 5 shows effective throughputs with drop-arrivals and with drop-from-queue when TCP Reno is used.

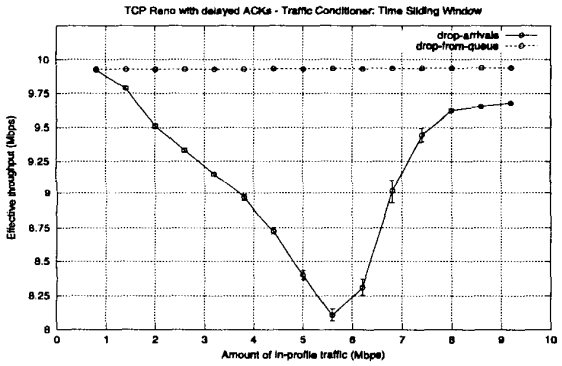

Figure 5 Throughputs – TCP R. and St. WRED.

With drop-arrivals, the effective throughput decreases as the amount of *in*-profile traffic increases up to 5.6 Mbps (Figure 5). Above this amount of *in*-profile traffic, throughput increases and approaches the effective throughput achieved with drop-from-queue. This confirms that TCP goes into slow-start more often with drop-arrivals than with drop-from-queue.

Bursty loss patterns increase the risk for dropping multiple packets from a TCP source within one window of data. When a queuing mechanism such as WRED is used to make drop decisions, *out*-of-profile packets are dropped in bursts when the average queue length (i.e., avg_ql) exceeds max_th(1). Figure 6 shows avg_ql with drop-arrivals and drop-from-queue with TCP Reno.

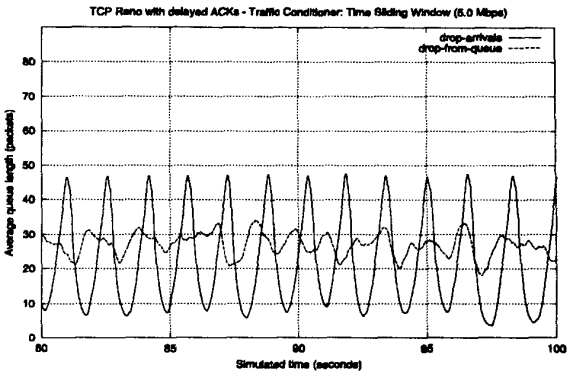

Figure 6 Queues – TCP Reno and St. WRED.

It can be seen in Figure 6 that the drop-arrivals strategy gives larger queue oscillation than the drop-from-queue strategy. Moreover, avg_ql exceeds max_th(1) frequently (max_th(1) is set equal to 45 packets). As mentioned above, this causes bursty loss patterns for *out*-of-profile traffic. Moreover, these bursts cause higher loss-rates than the smoother loss patterns obtained with drop-from-queue (i.e., although the long-term average queue lengths are similar for both drop strategies, the drop probability of 1 when avg_ql exceeds max_th(1) cause higher loss-rates).

3.3. TCP Sack and Standard WRED

For the simulations presented in this section, all sources use the Sack option of TCP. The differentiating queue mechanism is standard WRED. Figure 7 shows loss-rates for the drop-arrivals and the drop-from-queue strategy.

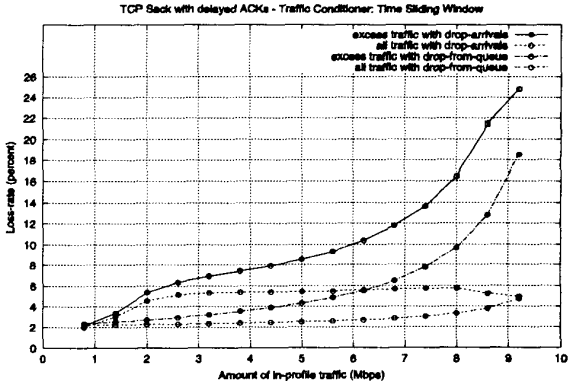

Figure 7 Loss-rates – TCP Sack and St. WRED.

With drop-from-queue, loss-rates are similar for TCP Reno and TCP Sack (Figure 4 and Figure 7). However, with drop-arrivals, loss-rates of *out*-of-profile traffic are

higher for most simulated amounts of *in*-profile traffic when TCP Sack is used. With TCP Reno, loss-rates are lower with drop-arrivals than with drop-from-queue at high amounts of *in*-profile traffic (Figure 4).

The Sack option enables TCP to better handle multiple losses within one window of data. Consequently, at bursty loss patterns, TCP Sack goes into slow-start more seldom than TCP Reno. Therefore, TCP Sack generates higher loads than TCP Reno with drop-arrivals. However, higher loss-rates with drop-arrivals than with drop-from-queue can still cause degradations in effective throughput. Figure 8 shows effective throughputs with drop-arrivals and with drop-from-queue when TCP Sack is used.

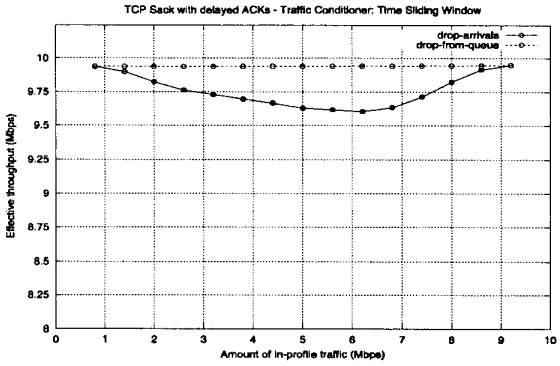

Figure 8 Throughputs – TCP Sack and St. WRED.

With drop-arrivals, the effective throughput gets degraded for most amounts of *in*-profile traffic (Figure 8). However, by comparing Figure 5 and Figure 8, it can be seen that the degradation is less severe with TCP Sack than with TCP Reno. This is because TCP Sack can better handle multiple losses within one window of data.

Since TCP Sack seldom goes into slow-start, queue oscillations can be expected to be less with TCP Sack than with TCP Reno. Figure 9 shows avg_ql with drop-arrivals and with drop-from-queue when TCP Sack is used by the traffic sources.

By comparing Figure 6 and Figure 9, it can be seen that with drop-arrivals less queue oscillation is obtained with TCP Sack than with TCP Reno. The queue oscillation is however still larger with drop-arrivals than with drop-from-queue. Moreover, as with drop-arrivals and TCP Reno, avg_ql exceeds max_th(1) frequently. Loss patterns are therefore burstier with drop-arrivals than with drop-from-queue also with TCP Sack, which cause higher loss-rates and lower link utilization than the smoother loss patterns obtained with drop-from-queue.

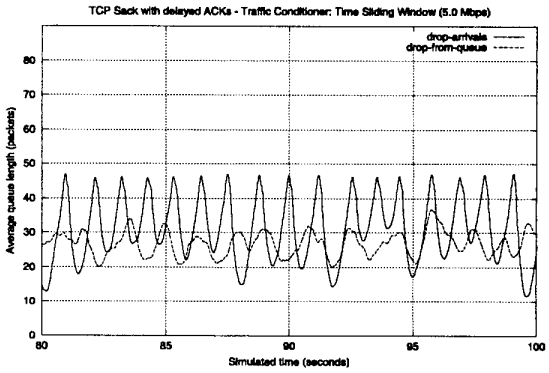

Figure 9 Queues – TCP Sack and St. WRED.

3.4. TCP Reno and Gentle WRED

For the simulations presented in this section, all sources use the Reno version of TCP. The differentiating queue mechanism is WRED with the gentle modification. Figure 10 shows loss-rates with drop-arrivals and with drop-from-queue.

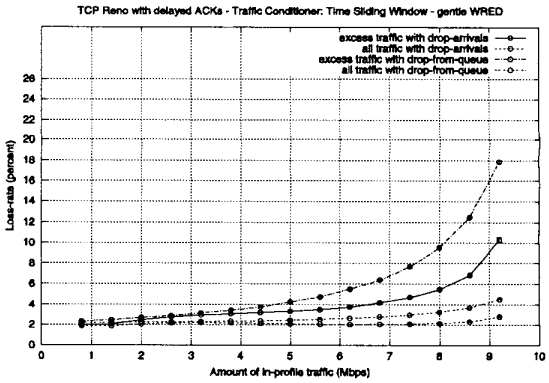

Figure 10 Loss-rates – TCP Reno and G. WRED.

With TCP Reno, loss-rates are higher with drop-from-queue than with drop-arrivals (Figure 10). This differs from results presented in Sections 3.2 and 3.3, for which loss-rates generally are higher with drop-arrivals than with drop-from-queue. The reason for the inverse relation in loss-rates is that loss patterns are burstier with drop-arrivals than with drop-from-queue. Hence, with drop-arrivals, the TCP Reno generate less traffic load than with drop-from-queue. This gives lower loss-rates.

With TCP Reno, burstier loss patterns can also result in decreased effective throughput. Figure 11 shows effective throughputs with drop-arrivals and with drop-from-queue respectively. TCP Reno and the gentle modification of WRED are used for these simulations.

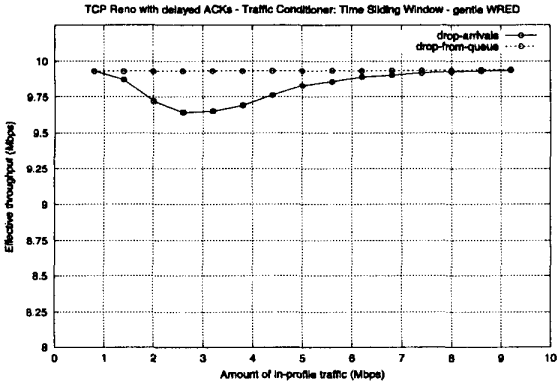

Figure 11 Throughputs – TCP R. and G. WRED.

Although not as much as for previous simulations (Figure 5 and Figure 8), effective throughput is decreased with drop-arrivals (Figure 11). This is because loss-rates are lower with the gentle modification. Loss patterns are also less bursty since loss probability does not instantaneously go from max_p(1) to 1 at max_th(1) with gentle WRED.

Since TCP Reno goes into slow-start more often with drop-arrivals than with drop-from-queue, queue oscillation can be expected to be larger with drop-arrivals. Figure 12 shows avg_ql with drop-arrivals and drop-from-queue with TCP Reno and gentle WRED.

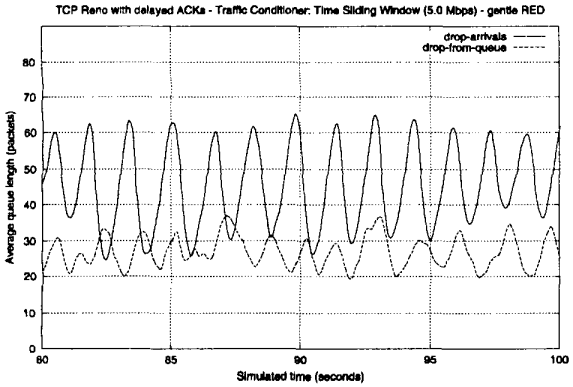

Figure 12 Queues – TCP Reno and Gentle WRED.

In Figure 12, it can be seen that queue oscillation is larger with drop-arrivals than with drop-from-queue. Moreover, the long-term average queue length is longer with drop-arrivals. Since the router needs to postpone drops with drop-arrivals, average queue lengths can be long although loss-rates are low (i.e., when drops are delayed, average queue length needs be longer to give the same loss-rates that immediate drops give). Large queue oscillation give bursty loss patters.

3.5. TCP Sack and Gentle WRED

For the simulations presented in this section, all sources use the Sack option of TCP. The differentiating queue mechanism is WRED with the gentle modification. Figure 13 shows loss-rates for the drop-arrivals and the drop-from-queue strategy.

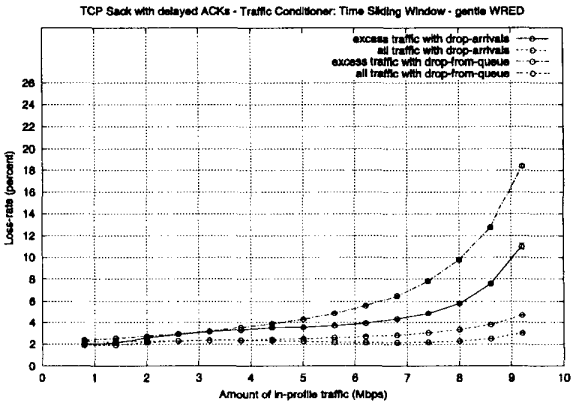

Figure 13 Loss-rates – TCP Sack and G. WRED.

The loss-rates presented in Figure 13 are similar to those presented in Figure 10. Compared to with TCP Reno, loss-rates are only a few percent higher at large amounts of *in*-profile traffic with TCP Sack. The higher loss-rates are caused by the higher loads generated by TCP Sack (i.e., due to fewer slow-starts).

Loss patterns can still be bursty with drop-arrivals though. Fewer slow-starts should give high effective throughputs, even with bursty loss patterns. Figure 14 shows effective throughputs for the drop-arrivals and the drop-from-queue strategy with TCP Sack and the gentle modification of WRED.

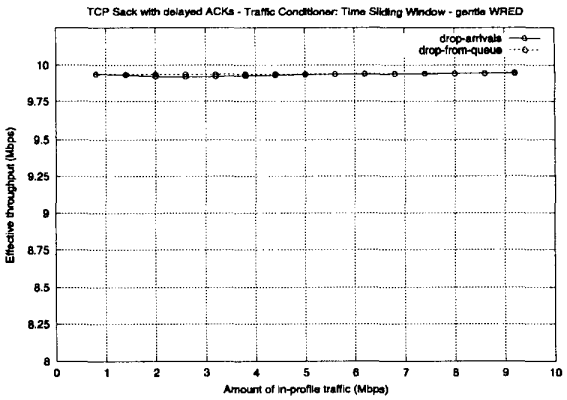

Figure 14 Throughputs – TCP S. and G. WRED.

As expected, effective throughputs are similar with drop-arrivals and drop-from-queue (Figure 14). Given this, the long-term average queue length should be similar as with TCP Reno (Figure 12), but queue oscillation should be less for TCP Sack. Figure 15 shows avg_ql with drop-arrivals and drop-from-queue with TCP Sack and gentle WRED.

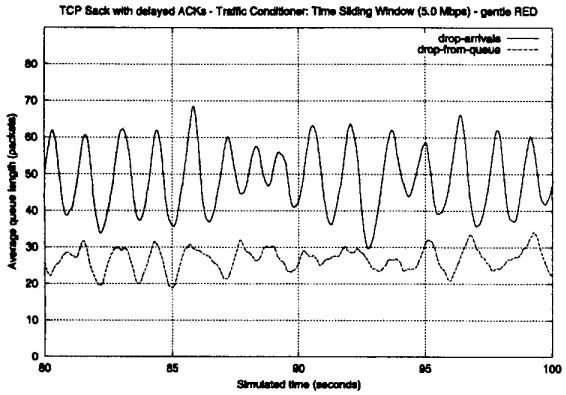

Figure 15 Queues – TCP Sack and Gentle WRED.

It can be seen in Figure 15 that queue oscillation is larger with drop-arrivals than with drop-from-queue. In addition, the long-term average queue length is about twice as long with drop-arrivals. This gives more jitter and longer delays with drop-arrivals compared to with drop-from-queue.

3.6. Summary of Simulation Results

The simulations presented show that less queue oscillation is obtained with drop-from-queue than with drop-arrivals. With standard WRED, large queue oscillation gives high loss-rates and bursty loss patterns. This forces TCP Reno into slow-start frequently. With a limited number of TCP sources, this can result in decreased link utilization. TCP Sack can better handle bursty loss patterns. Link utilization can however still be decreased due to higher loss-rates and that these TCP sources go into slow-start more often than they would with smoother loss patterns.

With the gentle modification of WRED, lower loss-rates are obtained with drop-arrivals than with drop-from-queue. This is because loss patterns are less bursty for the drop-from-queue strategy (i.e., due to less queue oscillation). Less bursty loss patterns enable the TCP sources to generate higher loads. With TCP Reno, the bursty loss patterns can cause decreased link utilization (as with standard WRED).

The lower loss-rates with gentle WRED come with longer queue lengths. The long-term average queue length is in our simulations about twice as long with drop-arrivals compared to with drop-from-queue (when half the link capacity is used by traffic at the lower drop precedence level). Long queue lengths and large queue oscillation means long queue delay with large variations (i.e., delay jitter).

4. Conclusions

In this paper, we show that packet loss and delay at congested links are influenced by actions taken to create loss-rate differentiation. In particular, the drop strategy used influences these quality metrics. A common drop strategy is to drop packets only as they arrive. We show, through simulations, that by dropping packets from the queue less queue oscillation is achieved than achieved by drooping packets only as they arrive. With standard WRED, less queue oscillation results in lower loss-rates and less bursty loss patterns.

The gentle modification proposed for RED can be applied to WRED. With gentle WRED, the problem of high loss-rates with drop-arrivals is alleviated. The queue oscillation is however still larger and loss patterns burstier when packets are dropped only as they arrive than when they are dropped from the queue. Moreover, the gentle modification increases the long-term average queue length. Long queue lengths and large queue oscillation means long queue delay with large variations (i.e., jitter).

Our conclusion is that dropping packets from the queue is preferable to dropping packets only as they arrive. The simulations indicate that this conclusion holds for both TCP Reno and Sack. Moreover, it holds for standard and gentle WRED as well as for TSW and token bucket based traffic conditioners[6].

5. References

[1] Black D. et. al., An Architecture for Differentiated Services, IETF RFC 2475, December 1998.

[2] Nichols K. et. al., Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, IETF RFC 2474, December 1998.

[3] Clark D. and Fang W., *Explicit allocation of best-effort packet delivery service*, IEEE/ACM Transactions on Networking, Volume 6, No. 4, pp. 362 – 373, August 1998.

[6] We have made simulations with pure token bucket and CAR [9] based traffic conditioners as well as with the TSW traffic conditioner. As these simulations show similar results as with the TSW traffic conditioner, these simulations are not included in the paper.

[4] Technical Specification from Cisco,
Distributed Weighted Random Early Detection, URL:
http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf.

[5] Floyd S. and Jacobson V., *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, August 1993.

[6] Braden B. et al, Recommendations on Queue Management and Congestion Avoidance in the Internet, IETF RFC2309, April 1998.

[7] UCB/LBNL/VINT Network Simulator - ns (version 2), URL: http://www-mash.cs.berkeley.edu/ns/.

[8] Hainanen J. et. al., *Assured Forwarding PHB Group*, IETF RFC 2597, June 1999.

[9] Technical Specification from Cisco, *Committed Access Rate (CAR)*, URL: http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/car.pdf.

[10] Bodin U., Schelén O. and Pink S. (2000), *Load-tolerant Differentiation with Active Queue Management*, ACM Computer Communications Review, Volume 30, Number 3, July 2000.

[11] Mathis M., Mahdavi J., Floyd S. and Romanow A, *TCP Selective Acknowledgment Options*, IETF RFC 2018, October 1996.

[12] Floyd, S., *Recommendation on using the "gentle_" variant of RED*, March 2000, URL:
http://www.aciri.org/floyd/red/gentle.html.

[13] Feldmann A., Gilbert A. C. and Willinger W., *Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control*, SIGCOMM'99, September 1999.

[14] Fong S. and Singh S., *Improving the Performance of Shared-Buffer Multistage ATM Switches under Bursty Traffic using Backpressure and Pushout*, ICCCN `96, October 1996.

[15] Fong S. and Singh S, *Modelling and Performance Analysis of Shared Buffer ATM Switches with HotSpot Pushout under Bursty Traffic*, Australian Telecommunication and Network Application Conference, Vol.2, pp.561-566, Melbourne, December 1996.

[16] Chao H. J., Cheng H., Jenq Y-R. and Jeong D., *Design of a Generalized Priority Queue Manager for ATM Switches*, IEEE Journal of Selected Areas in Communications, Vol. 15, No. 5, pp. 867-880, June 1997.

[17] Choudhury A. K. and Hahne E. L., New Implementation of Multi-Priority Pushout for Shared Memory ATM Switches, Computer Communications, Vol. 19, No. 3, pp. 245-256, March 1996.