Load-tolerant Differentiation with Active Queue Management

Ulf Bodin, Olov Schelén⊗ and Stephen Pink*
Computer Science and Electrical Engineering
Luleå University of Technology, CDT, SE - 971 87 Luleå, Sweden
⊗also at Swedish Institute of Computer Science (SICS), Kista, Sweden
**also at Telia Research AB, Luleå, Sweden*
{uffe, olov, steve}@cdt.luth.se

Abstract

Current work in the IETF aims at providing service differentiation on the Internet. One proposal is to provide loss differentiation by assigning levels of drop precedence to IP packets. In this paper, we evaluate the active queue management (AQM) mechanisms RED In and Out (RIO) and Weighted RED (WRED) in providing levels of drop precedence under different loads. For low drop precedence traffic, RIO and WRED can be configured to offer sheltering (i.e., low drop precedence traffic is protected from losses caused by higher drop precedence traffic). However, if traffic control fails or is inaccurate, such configurations can cause starvation of traffic at high drop precedence levels. Configuring WRED to instead offer relative differentiation can eliminate the risk of starvation. However, WRED cannot, without reconfiguration, both offer sheltering when low drop precedence traffic is properly controlled and avoid starvation at overload of low drop precedence traffic. To achieve this, we propose a new AQM mechanism, WRED with Thresholds (WRT). The benefit of WRT is that, without reconfiguration, it offers sheltering when low drop precedence traffic is properly controlled and relative differentiation otherwise. We present simulations showing that WRT has these properties.

1. Introduction

The traditional Internet architecture offers best-effort service only. The Internet community has recognized the importance of simplicity in forwarding mechanisms, but also that a single service may not be enough to support the wide range of applications on the Internet. The Internet Engineering Task Force (IETF) is therefore designing architectural extensions to enable service differentiation on the Internet. The Differentiated Services (DiffServ) architecture [1][2] includes mechanisms for differentiated forwarding.

One proposed mechanism for DiffServ is to assign levels of drop precedence to IP packets. This mechanism is included in the Assured Forwarding (AF) per-hop behavior (PHB) group [12]. AF can be used to offer differentiation among rate adaptive applications that respond to packet loss, 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. In addition, a packet within a user's profile may be tagged with one out of several levels of drop precedence. For now, there are three levels of drop precedence specified for AF.

When creating differentiation with levels of drop precedence, packets within an application data stream may get tagged with different drop precedence levels depending on whether they are considered *in* or *out* of profile. For AF, it is required that packets within an application data stream tagged with different drop precedence levels are not reordered by routers. Packet reordering can reduce the performance of TCP and real-time applications using UDP.

Moreover, for AF, it is required that the levels of drop precedence are ordered so that for levels $x < y < z$, $P_{drop}(x) < P_{drop}(y) <= P_{drop}(z)$[1]. Within this order, AF leaves freedom in further tuning drop precedence probabilities. For example, drops can be strictly given to high precedence traffic so that $P_{drop}(z)$ approaches 1 before any packets at other levels are dropped, or drop probabilities can be relatively distributed among precedence levels, etc. To characterize queuing mechanisms offering multiple levels of drop precedence, we introduce two properties, *sheltering* and *load-tolerance*.

We denote a drop precedence level as sheltered if traffic loads at higher precedence levels only have minor effects on the loss-rate experienced by traffic at this level. The *sheltering* property holds for a queuing mechanism if it offers such protection for traffic at one or more precedence levels. Sheltering is justified by requirements for predictability. When sheltering is provided, the network can be provisioned and traffic profiles can be defined to offer users a predictable service for their *in* traffic. As *in* traffic is sheltered, the aggregated amount of *out* traffic in the network will only have minor effects on the predictability of such service. However, if traffic control fails or is inaccurate, sheltering can cause starvation of higher precedence traffic.

The *load-tolerance* property holds for a differentiating queuing mechanism if it meets the following two requirements at overload:

- *Prevent starvation of high drop precedence traffic.*
  High drop precedence traffic must always get a useful share of the bandwidth available (i.e., even if low drop precedence traffic is not properly controlled).
- *Preserve hierarchy among drop precedence levels.*
  Traffic at a drop precedence level must always experience less drop probability than traffic at a higher drop precedence level. As mentioned above, this is also a requirement for AF.

[1] $P_{drop}(x)$ is the drop probability for traffic at precedence level x.

Load-tolerance can be justified by recommendations for the DiffServ architecture. Preventing long-term starvation of best-effort traffic (normally given the Default PHB) is advocated in [2]. The DiffServ architecture allows packets initially tagged for the Default PHB to be re-tagged with another PHB. Re-tagging best-effort traffic with a high drop precedence level within an AF class makes it possible to explicitly control the relation in treatment between *out* traffic and best-effort traffic. This is appealing since prioritized traffic (*in* and *out* traffic together) can then be guaranteed equal or better treatment than best-effort traffic. If best-effort traffic would be forwarded in a separate queue to avoid starvation, such a guarantee is harder to provide.

Services sheltering *in* traffic and guaranteeing equal or better treatment than best-effort can be constructed with a differentiating queuing mechanism that does not by itself prevent starvation. Then, to protect best-effort traffic against long-term starvation, one must however rely on accurate control of *in* traffic.

In the context of DiffServ, traffic control based on either dynamic admission control or statistically allocated service profiles has been discussed. Dynamic admission control is likely to be adequate in protecting best-effort traffic against long-term starvation. It may not however protect against transient starvation. This is because the traffic control may fail due to inaccuracies in admission control and topology changes. Some ISPs may accept transient starvation, but others may consider it important to avoid.

For statistically allocated, destination-independent, service profiles, longer periods of overload may be encountered at topology changes or for destinations that suddenly become more attractive than expected. Consequently, traffic control based on such service profiles may not be adequate in protecting best-effort traffic against either long-term or transient starvation.

The objective of this work is to show that a drop differentiating queuing mechanism can be designed to offer sheltering if *in* traffic is properly controlled (i.e., *conditional* sheltering) and to meet our requirements for load-tolerance if traffic control fails or is inaccurate. With these properties, traffic control need not be accurate to protect high precedence traffic against long-term starvation. Statistically allocated, destination-independent, service profiles can then be used without risking long-term or transient starvation of high precedence traffic. Moreover, conditional sheltering and load-tolerance can be appealing to avoid transient starvation when dynamic admission control is used.

Multiple levels of drop precedence can be created with an AQM mechanism applied to a FIFO queue. An appealing property of FIFO queues is that packets are forwarded in the same order as they arrive. Thus, packet reordering is avoided. Moreover, FIFO queues are suitable for high-speed links since they can be implemented efficiently.

In this paper we evaluate the appropriateness of two AQM mechanisms, RIO [4] and WRED [5], in providing sheltering under different loads. In section 3.3, we show that RIO and WRED can be configured to offer sheltering. Then, however, these mechanisms can cause starvation of higher drop precedence traffic if there is an overload of low drop precedence traffic, i.e., with such configuration they cannot meet our requirements for load-tolerance.

In section 3.4.1, we show that WRED can meet our requirements for load-tolerance when configured to offer a relative differentiation. Relative differentiation means, in this context, that traffic at a drop precedence level experiences a loss-rate defined in relation to the loss-rate experienced by traffic at another precedence level. Next, in section 3.4.2, we show that RIO can be configured to prevent starvation, but then a hierarchy among precedence levels cannot be guaranteed under periods of overload. That is, traffic tagged with a low drop precedence level may experience a larger loss-rate than traffic at a higher precedence level. Such a configuration of RIO is therefore not advisable.

Since neither RIO nor WRED can meet our requirements for load-tolerance when providing sheltering, we propose a new AQM mechanism, WRED with Thresholds (WRT). The benefit of WRT is that, without reconfiguration, it offers sheltering if low drop precedence traffic is properly controlled and relative differentiation otherwise. Thus, WRT meets our requirements for load-tolerance. We examine the load-tolerance of WRT through simulations. With these simulations, WRT is compared with RIO and WRED to show that WRT can offer the same differentiation as these mechanisms. Moreover, simulations evaluating properties of WRT when used to construct services are provided.

Load-tolerant and conditional sheltering queuing mechanisms are appealing for constructing predictable end-to-end services that guarantee users equal or better service than users of the best-effort service. The load-tolerance property allows traffic control to be less conservative in considering rare network failures, as a relative differentiation is a minimum guarantee, and traffic with higher drop precedence will not be starved. Although services are discussed in this paper, we do not focus on construction of end-to-end services. Our main contribution is that we show how an AQM mechanism can be designed to provide both conditional sheltering and load-tolerance.

The rest of this paper is structured as follows: In section 2, related work is discussed. Section 3 discusses basic properties of AQM using Random Early Detection (RED) as an example. Then, the applicability of WRED and RIO for offering differentiable levels of drop precedence is discussed. In section 4, a new queue mechanism, WRT, is proposed. In section 5, we present simulations to evaluate the basic properties of WRT. Section 6 discusses some implications WRT might have on end-to-end service construction. Finally, in section 7 we summarize our major findings.

2. Related Work

Differentiation in IP networks can be created with queue management mechanisms, scheduling mechanisms using multiple queues, or combinations of those. Multiple queues may, however, cause packet reordering. As pointed out in section 1, reordering of packets within an application data stream should be avoided when creating differentiation between levels of drop precedence. For this reason, we do not consider multiple queue schemes in this paper.

WRED and RIO, which we evaluate in this paper, are two AQM mechanisms designed to offer multiple levels of drop precedence. Another AQM mechanism that could be extended to provide multiple levels of drop precedence is Fair RED (FRED) [18]. However, since FRED relies on per-flow information[2], it can be expected to need more memory and processing than simpler mechanisms such as WRED and RIO. For queuing mechanisms, memory consumption and processing cost should be minimized. In this paper, we present a new sheltering AQM mechanism and show that it meets our requirements for load-tolerance (as defined in section 1). This new mechanism does not use any per-flow information. Since we can provide the differentiation we are aiming for without per-flow information, we do not evaluate FRED in this paper.

Within the IETF, there are works in progress evaluating different issues of the DiffServ architecture with simulations. One issue studied is how to prevent unresponsive UDP traffic from getting more than its fair share of the unreserved bandwidth in an AF class [12]. In [14] and [15], the need for two or three levels of drop precedence to solve this issue is studied. Moreover, in [16] different assignments of three drop precedence levels to unresponsive UDP and responsive TCP traffic are studied to evaluate this issue of fairness. Another issue studied is how to achieve fairness among flows within an AF class. A new tagging algorithm, the fair marker, is presented and evaluated with simulations in [17].

None of the above-referred works study the aspect of load-tolerance, which this paper is focused on. Those works implicitly presume that traffic at a sheltered drop precedence level is forwarded separately from best-effort traffic. This can, for example, be achieved with separate queues for sheltered and best-effort traffic respectively. Some ISPs may prefer such solutions. However, forwarding best-effort traffic separately from prioritized traffic (*in* and *out* traffic together) implies that *out* traffic may get treated different to best-effort traffic. Moreover, with separate queues, this difference is hard to control. Our work shows that traffic at a sheltered precedence level can be forwarded in the same queue as best-effort traffic without risking starvation of high drop precedence traffic. This enables an explicit control of the relation in treatment between *out* traffic and best-effort traffic, which can be beneficial. For example, with an explicit control of this relation, prioritized traffic (*in* and *out* traffic together) can be guaranteed equal or better treatment than best-effort traffic.

[2] Per-flow states are kept for flows present in the queue.

3. Active Queue Management (AQM)

WRED and RIO are two AQM mechanisms designed to provide multiple levels of drop precedence. They are both extensions of RED. In this section, we describe RED (section 3.1), RIO and WRED (section 3.2). Section 3.3 discusses the risk of starvation when offering *sheltering* with WRED or RIO. In section 3.4, we examine these mechanisms' ability to meet our requirements for load-tolerance (as defined in section 1) when offering *relative* differentiation.

3.1 Random Early Detection (RED)

RED was originally proposed in 1993 by Floyd and Jacobson [9] and is now recommended for deployment in the Internet [10]. RED allows a router to drop packets before any queue becomes saturated. Consequently, congestion responsive flows will back-off early resulting in shorter average queue lengths. This is appealing for several reasons. First, the queuing delay will decrease, which is good for interactive applications. Second, packet drops will not occur in bursts. RED achieves this by dropping packets with a certain probability depending on the average queue length (avg_ql in Figure 1).

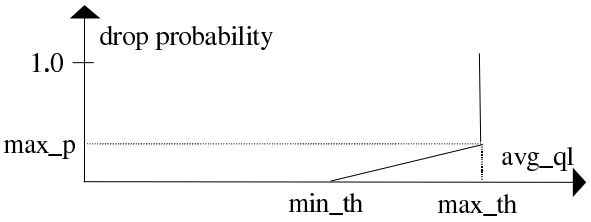

Figure 1 The RED mechanism.

3.2 Weighted RED and RED In and Out

WRED [5], defined and implemented by Cisco, and RIO, proposed and evaluated with simulations by Clark and Fang [4], are two AQM mechanisms defined for service differentiation in IP networks. They are both based on RED and offer differentiation by managing drop precedence.

With WRED, eight separate levels of drop precedence can be supported. Each of these levels is configured with a separate set of RED parameters (see Figure 2). RIO, on the other hand, has only two sets of RED parameters. Hence, in its basic version, two levels of drop preference are supported, i.e., one level for packets tagged as *in* profile and another level for packets tagged as *out* of profile.

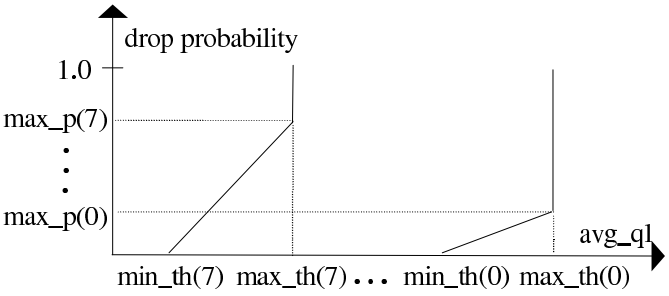

Figure 2 The WRED mechanism.

The main difference between WRED and RIO is that WRED uses *one* average queue length to calculate drop probabilities while RIO uses *two* average queue lengths. WRED calculates its average queue length (avg_ql) based on all packets present in the queue. RIO does that too but, in addition, it calculates a separate average queue length for packets in the queue tagged as *in* profile (avg_ql_in), see Figure 3.

Recommendations on how WRED and RIO should be parameterized can be found in [4] and [5] respectively. With the recommended setting of WRED, a relative differentiation is obtained. The recommended setting of RIO provides sheltering. We do not, however, stick to these recommendations in our evaluation of these mechanisms. To examine whether RIO or WRED can be configured to provide sheltering and meet our requirements for load-tolerance, we need to consider any possible configuration of these mechanisms.

The parameter settings given in sections 3.3 and 3.4, where creations of sheltered and relative differentiation respectively are discussed should be seen as rough recommendations. As pointed out in [13], finding an optimal RED configuration is non-trivial.

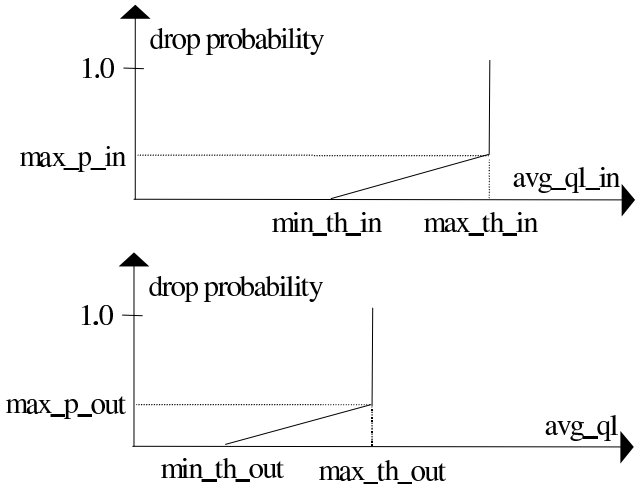

Figure 3 The RIO mechanism.

3.3 Creating Sheltering

We denote a drop precedence level as *sheltered* if traffic loads at higher precedence levels only have minor effects on the loss-rate experienced by traffic at this level. Hence, this loss-rate can be limited by controlling traffic at the sheltered level only (i.e., traffic at other drop precedence levels need not be controlled).

In this section, we show that neither WRED nor RIO can meet our requirements for load-tolerance when configured to provide sheltering. With these mechanisms, traffic at a sheltered level has to be properly controlled to avoid starvation of higher drop precedence traffic and to ensure that a hierarchy is preserved between precedence levels.

3.3.1 Sheltering with WRED

With WRED, sheltering is offered with parameter settings satisfying the following two rules:

(1) $\max_th(n) < \max_th(n-1)$, and

(2) $\max_th(n) < \min_th(n-1)$ $(n = 1, \ldots, 7)$

where lower *n* means lower drop precedence.

The setting of the max_p(#)s does not affect the sheltering. A configuration satisfying rules (1) and (2) is shown in Figure 2. Satisfying these rules is needed to prevent uncontrolled traffic at higher drop precedence levels from causing more than occasional losses to traffic at lower precedence levels. For example, uncontrolled traffic at precedence level *one* can be expected to cause avg_ql to exceed max_th(1) with a few packets (or bytes) for short periods. Hence, setting max_th(0) and/or $\min_th(0) \leq \max_th(1)$ would allow traffic at precedence level *one* to cause more than occasional losses to traffic at precedence level *zero*. This would break the sheltering of precedence level *zero*.

With a configuration satisfying rules (1) and (2), sheltering of traffic at precedence levels 0 to 6 is offered. However, to avoid starvation of traffic at drop precedence level 7, avg_ql must not exceed max_th(7) for any longer period (which must be ensured by traffic control applied to traffic at precedence levels 0 to 6)[3]. In general, to prevent starvation of traffic at drop precedence level n+1, traffic control applied to traffic at drop precedence level n must ensure that avg_ql does not exceed max_th(n+1) ($n = 0, \ldots, 6$). Hence, WRED cannot meet our first requirement for load-tolerance when configured to provide sheltering.

3.3.2 Sheltering with RIO

Since RIO uses a separate average queue length for packets tagged as *in* profile, it offers sheltering with any configuration. However, max_th_in should be set equal to or larger than max_th_out. Otherwise, traffic tagged as *in* profile may experience a higher loss-rate than traffic tagged as *out* of profile. This would break the hierarchy between precedence levels (i.e., if the level for *in* packets is to provide lower drop precedence than the level for *out* packets). Such configuration is not advisable since it cannot meet our second requirement for load-tolerance, which also is a requirement for AF.

The configuration of RIO shown in Figure 3 offers sheltering and preserves the hierarchy. With this kind of configuration, starvation of high drop precedence traffic (*out* traffic) can however occur if low precedence traffic (*in* traffic) is not properly controlled. With RIO, this control has to ensure that avg_ql does not exceed max_th_out for any longer period. Consequently, RIO cannot meet our first requirement for load-tolerance when configured to preserve a hierarchy between drop precedence levels.

[3] Traffic at precedence level 0 may cause avg_ql to exceed max_th0 with a few packets (or bytes). That would however not cause starvation of that traffic since avg_ql will shrink below max_th(0) when packets at that precedence level get dropped.

3.4 Creating Relative Differentiation

We consider two precedence levels to be *relative* differentiated when traffic at these levels experiences a definable relation in loss-rates. If, say, R_i is the loss-rate offered by drop precedence level i and R_j is the loss-rate offered by drop precedence level j when traffic is present at both these levels. Then the relation in loss-rates between precedence levels i and j, for $i < j$, can be specified as:

(3) $R_i < k + l * R_j$

or as:

(4) $R_i = k + l * R_j \qquad (i, j = 1 \ldots N)$

where k and l are constants.

WRED can meet the requirements for load-tolerance when configured to offer relative differentiation among drop precedence levels. This is described in section 3.4.1. Section 3.4.2 shows that RIO is unable to offer relative differentiation.

3.4.1 Relative Differentiation with WRED

Relative differentiation is offered by WRED if all max_th(#)s are set equally. The differentiation offered then depends on the settings of min_th(#)s and max_p(#)s. These parameters should be set to ensure a hierarchy between the levels of drop precedence. That is, traffic at low drop precedence levels should, at any average queue length longer than min_th(7), experience lower drop probability than traffic at higher precedence levels.

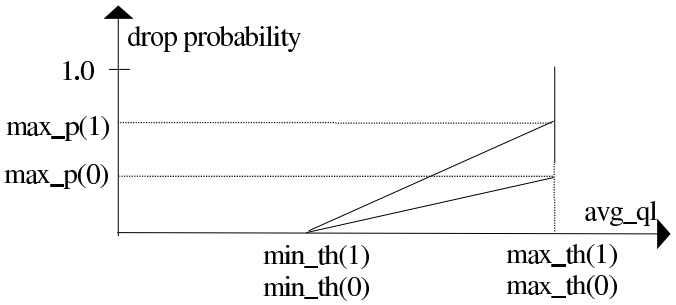

Figure 4 WRED configured to offer a relative differentiation.

With the setting of WRED shown in Figure 4, traffic at precedence level *zero* will, when avg_ql exceeds the min_th(#)s, experience less loss-rate compared to traffic at precedence level *one* in times of congestion. An exact relation in loss-rates between traffic at different levels of drop precedence cannot however be guaranteed with WRED (i.e., the difference in loss-rates can be larger or less than expected). This is because WRED uses the average queue length (avg_ql) to differentiate between precedence levels. This variable will vary over time with the arrival-rate of packets. For example, if traffic at a drop precedence level is burstier than traffic at other precedence levels. Then the relation in loss-rates between that level and higher drop precedence levels can be less than expected. Depending on traffic characteristics, the relation in loss-rates between a pair of drop precedence levels can also be larger than expected.

An approach to improve the predictability of relations in loss-rates is to keep backlogs on previous drops for each precedence level. This approach is described and evaluated in [19].

3.4.2 Relative Differentiation with RIO

RIO cannot be configured to offer relative differentiation. This is because RIO uses a separate variable (avg_ql_in) to calculate the probability, P_{in}, of dropping an arriving packet tagged as *in* profile. This separate variable does not contain any information about the amount of packets tagged as *out* of profile present in the queue. The calculation of P_{in} can therefore not be related with the probability P_{out} of dropping the packet if it had been tagged as *out* of profile.

4. Definition of a Load-tolerant AQM Mechanism

In this section, we define a new AQM mechanism that, without reconfiguration, offers sheltering when low drop precedence traffic is properly controlled and relative differentiation otherwise. This new mechanism, Weighted RED with Thresholds (WRT), is designed by combining RIO with WRED. However, before presenting WRT, we define another mechanism, named load-tolerant RIO (ltRIO), which is a special case of WRT.

4.1 Definition of Load-tolerant RIO (ltRIO)

We adopt, from RIO, the idea of calculating two separate average queue lengths. However, instead of discarding packets tagged as *in* profile when avg_ql_in exceeds max_th_in (Figure 3), these packets are treated as if they were tagged as *out* of profile. When avg_ql_in exceeds th_in, we use avg_ql (i.e., the average queue length for all packets present in the queue) to make drop decisions for *in* packets. Note that a decision of treating *in* packets as if they were tagged as *out* of profile is for one queue only. That is, packets tagged as *in* profile are not re-tagged as *out* of profile.

To avoid starvation, max_th_in must be set lower than max_th_out. This configuration provides sheltering as long as avg_ql_in does not exceed min_th_in. If avg_ql_in does exceed max_th_in, the AQM mechanism will behave as RED (i.e., there will be no differentiation).

Note that, for ltRIO, calculating a separate average queue length for *in* packets is necessary to shelter those packets from packets tagged as *out* of profile. That is, to ensure that *in* packets do not suffer from more than occasional packet losses caused by overload of traffic tagged as *out* of profile.

4.2 Definition of WRED with Thresholds (WRT)

With ltRIO, packets are dropped using RED parameters coupled to the average queue length for both *in* and *out* packets in the queue. Thus, to perform random congestion signaling, RED parameters coupled to avg_ql_in are not necessarily needed. At overload, these parameters can however be used to perform random *early* congestion signaling for *in* traffic.

By setting min_th_in equal to max_th_in, *in* traffic get protected from more than occasional losses caused by overload of *out* traffic as long as avg_ql_in does not exceed these thresholds. This is appealing when constructing services guaranteeing *in* traffic very low loss-rates. Since we consider such services, we chose to reduce the number of parameters present in ltRIO by using a single threshold instead of a set of RED parameters to make drop decisions for *in* traffic.

Since ltRIO does not offer any differentiation when avg_ql_in exceeds max_th_in, the hierarchy among precedence levels may get broken. One way to preserve the hierarchy when this happens is to switch from sheltered to relative differentiation.

As discussed in the previous section, WRED provides relative differentiation when all max_th(#)s are set equally. Thus, we combine ltRIO with WRED to get this property in our new AQM mechanism. For this mechanism, we do not allow the max_th(#)s to be set separately from each other. This is because such setting may cause starvation of traffic at high drop precedence levels.

The combined scheme, WRT (Figure 5), provides relative differentiation between N levels of drop precedence when avg_ql_in exceeds th_in. That is, when avg_ql_in exceeds th_in, avg_ql is (as for ltRIO) used to make drop decisions for *in* packets. However, in contrast to ltRIO, differently tagged *in* packets can be relatively treated to each other and to *out* packet. The relative differentiation is configured with the min_th(#)s and the max_p(#)s parameters.

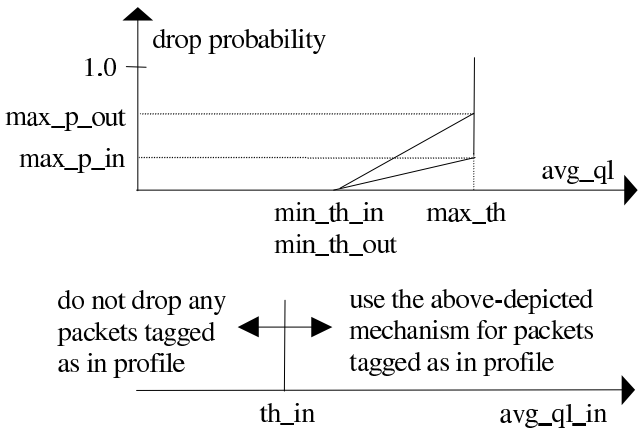

Figure 5 The WRT mechanism.

```
for each packet arrival
   calculate avg_ql and avg_ql_in;
   if the packet is tagged as in
      if avg_ql_in > th_in
         if min_th_in < avg_ql < max_th
            calculate probability Pin;
            with Pin, drop this packet
         else if avg_ql >= max_th
            drop this packet
   else if the packet is tagged as out
      if min_th_out < avg_ql < max_th
         calculate probability Pout;
         with Pout, drop this packet
      else if avg_ql >= max_th
         drop this packet
```

Figure 6 The pseudo-code of WRT.

In this paper we only use two precedence levels, which are called the *in* and *out* level respectively. Figure 6 shows how WRT, with two levels of drop precedence, can be implemented. The implementation has basically the same complexity as an implementation of RIO[4].

Whenever needed, WRT can be extended to support more levels of drop precedence. Hence, WRT can be used to support the AF PHB group in the IETF DiffServ framework. For AF, three levels of drop precedence are specified [12]. To support three precedence levels, WRT can be extended with one more threshold associated with an additional average queue length. In this case, the average queue length for the lowest precedence level is calculated based on packets tagged with that level only. Next, the average queue length for the middle level is calculated based on packets tagged with that level and packets tagged with the lowest precedence level. Finally, an average queue length is calculated for all packets present in the queue.

When the average queue length for the lowest precedence level exceeds the threshold associated with this level, packets at this level are treated as if they were tagged with the middle level. When both the average queue length for the lowest level and the average queue length for the middle level exceeds their thresholds, a relative differentiation between the three precedence levels is provided. The relative differentiation depends on how the min_th(#) and max_p(#) are configured for each of these levels and the current traffic load.

The threshold for the lowest precedence level must be set to a equal or lower value than the threshold for the middle level. This is because the order at which the thresholds are set defines the order in priority between the precedence levels. We recommend the following configuration rules for a WRT queue with N levels of drop precedence:

- $th_(0) \le th_(1) \le \ldots \le th_(N-1) < max_th$
- $max_p(0) < max_p(1) < \ldots < max_p(N)$
- all min_th(#)s set equally and larger than th_(N-1)

The first two rules are to achieve a hierarchy between precedence levels. Setting all min_th(#)s equally creates a relative differentiation that offers a fixed relation for any avg_ql between the value of these parameters and max_th. Moreover, when low precedence traffic is properly controlled, setting min_th(#)s larger than th_1 gives traffic at precedence level *zero* a queue space equal to min_th(#)s – th_1 before any packet at that level has to be dropped. Hence, that traffic can be given a useful share of the bandwidth.

5. Simulations

In this section, we present simulations testing the load-tolerance of ltRIO and WRT. The simulations are made with the *network simulator* (ns) [11]. The simulation setup is described in section 5.1. Using this setup, we validate that ltRIO can offer the same differentiation as RIO and that WRT can offer the same differentiation as WRED (section 5.2).

[4] An efficient implementation of an AQM mechanism should use integer arithmetic and a background process to make operations that do not need to be made as part of the forwarding process.

To evaluate the behavior of these mechanisms when constructing services, we study how the differentiation offered depends on the amount of traffic tagged as *in* profile (section 5.3). Next, we identify, for a specific traffic load, topology and two configurations, the maximum load of *in* traffic for which sheltering can be preserved (section 5.4). Thereafter, we study the behavior of RIO, ltRIO and WRT when the load of *in* traffic is gradually increased to exceed the maximum load of *in* traffic for one of these configurations (section 5.5). Finally, we summarize the simulations (section 5.6).

5.1 Simulation Setup

In the simulations, a simple topology with ten hosts (S0, …, S9) connecting to their respective destinations (R0, …, R9) via one common link is used. This link is a bottleneck of 30 Mbps with 20 ms delay (Figure 7). The AQM mechanisms evaluated are applied to the queue attached to the bottleneck link. Each host has ten TCP Reno connections with their respective destination. The throughput for each of these TCP flows is measured over the time of 16 simulated seconds. Every simulation goes through an initiation phase of four simulated seconds before measurements are initiated. This is to let the queue stabilize before the behavior of the AQM mechanisms we want to evaluate is observed.

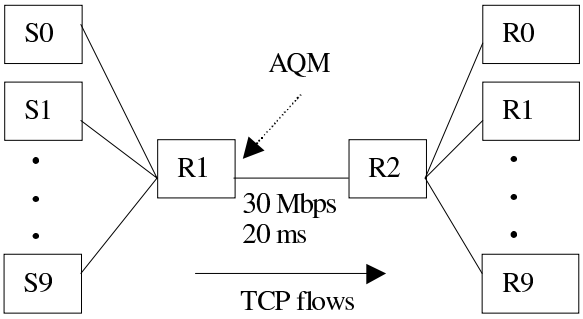

Figure 7 The simulation setup.

During the time when throughput is measured, the aggregate throughput is close to 30 Mbps in all simulations. The TCP connections are initiated randomly within the first simulated second. All these connections have the same RTT (40 ms plus queuing delay). With equal RTT for all connections, we avoid that connections with shorter RTT get higher throughput than connections with longer RTT. Such differences in throughput would make our evaluation of load-tolerance complicated.

Certainly, the traffic distribution used in our simulations does not correspond to what recently has been observed on the Internet. To create a more realistic traffic scenario, a large amount of short-lived TCP flows should be used in addition to long-lived TCP flows, some portion of UDP traffic will also be needed, and the amount of traffic present should be varied over time with some heavy-tailed statistical distribution. Such a scenario would however make the load vary randomly due to the statistics used. Our simple traffic scenario enables us to control the load more accurately, which is needed to evaluate load-tolerance.

A time sliding window (TSW) rate estimator [4] is used for each of the ten hosts to tag packets as *in* profile up to a certain rate. Thus, one service profile is applied for all ten TCP connections at every single host. The TSW rate estimator calculates, upon each packet arrival, the average rate for packets that have arrived over a period. By tagging packets as *out* of profile when the average rate exceeds a certain threshold, the burstiness of TCP packets tagged as *in* profile is smoothed out.

As discussed in [4], there are two different approaches to how packets can be tagged based on the rate estimated with TSW. The first approach is more general and can be applied to aggregated TCP traffic as well as to individual TCP connections. The second approach should only be applied to individual connections but is then more effective if the estimator is placed close to the sending host. Since we apply the estimator to an aggregate of ten TCP connections, the first approach is more appropriate for our simulations.

With the first approach, the TSW window size should be set to a large value (i.e., in the order of a TCP saw tooth from 66 to 133 percent of the rate specified in the service profile). This is recommended in [4]. Too large a TSW window, the traffic tagged as *in* profile can become burstier (e.g., bursts shorter than the TSW window size may not be detected and packets tagged as in profile may thus be burstier). On the other hand, too small a TSW window may cause the aggregate throughput to be less than what is specified in the profile. For example, if the rate at which an aggregate of TCP sources send packets varies with a period shorter than the TSW window. Then packets may be tagged as *out* of profile too often. This is because the rate estimated with a short TSW window varies more than it would with a larger TSW window. If those out packets get dropped in the network, the aggregate rate of the TCP sources may not reach the target rate.

Consequently, the TSW window size may affect the throughput experienced by individual TCP flows and thus the variation in arrival-rate of packets tagged as *in* profile. Unfortunately, this implies that there is a circular dependency between the length in time of a saw tooth and the TSW window size. In addition, the length of a saw tooth will vary because packets get randomly dropped in the network. An appropriate TSW window size for a certain TCP connection is therefore hard to choose based on known parameters only. Thus, it might be necessary to adapt the TSW window size based on real-time measurement of each individual TCP flow. We do not, however, evaluate the issue of adapting the TSW window size in this paper since it is focused on queuing mechanisms and not traffic conditioning.

For all our simulations, the window size is set to 300 ms. This value was chosen from the following calculations. Assume that the target rate of a certain TCP connection is set to 500 kbps. In our simulations, the RTT is 80 ms (including average queuing delay) and the average packet size is 8000 bits. This TCP connection will then on average have five packets on the fly and an

average congestion window of five packets of data[5]. Optimally, the number of packets on the fly and the size of the congestion window will then vary between 1.33 * 5 and 0.66 * 5. The variation is thus 0.67 * 5 = 3.35 packets. Since TCP increases its congestion window with at most one segment of data[6] for each RTT during congestion avoidance, the length in time of a TCP saw tooth is 3.35 * 0.08 = 0.268 s.

5.2 Properties of ltRIO and WRT

In this section, the properties of ltRIO and WRT are evaluated in comparison with RIO and WRED. Especially, we study the differentiation these mechanisms offer during overload of traffic tagged as *in* profile. To perform this evaluation, the average throughput experienced by TCP sources sending all their packets tagged as *in* profile is observed (i.e., these sources have unlimited rate profiles). This is compared with the average throughput experienced by other TCP sources sending all their packets tagged as *out* of profile (i.e., sources with zero-rate profiles). The fraction of TCP sources with unlimited rate profiles is varied between 10 and 90 percent in steps of 10. The results are plotted in graphs with the average throughput at the y-axis and the fraction of sources with unlimited rate profile at the x-axis. Figure 8 shows the results for WRED and RIO when configured to offer sheltering with the risk of starving high precedence traffic at overload (configuration 1 of WRED and RIO).

WRED is, in this simulation, configured with max_th(0) set to 200 packets, min_th(0) to 150 packets, max_th(1) to 100 packets, min_th(1) to 50 packets, max_p(0) and max_p(1) to 5 percent. The parameters for the other six precedence levels are not relevant since only level zero and one are used (level one is applied to traffic tagged as *in* profile and level zero to traffic tagged as *out* of profile). RIO is configured equally (i.e., max_th_in set to 200 packets, min_th_in to 150 packets, max_th_out to 100 packets, min_th_out to 50 packets, max_p_in and max_p_out to 5 percent).

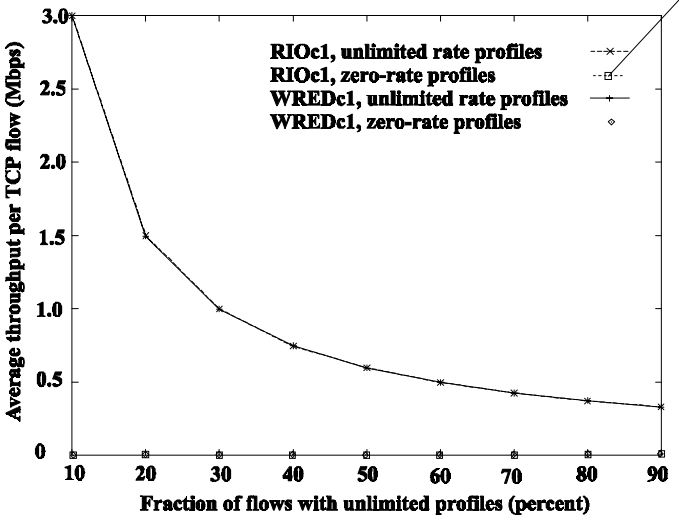

Figure 8 Throughput with WRED and RIO (configuration 1).

With the configuration used in Figure 8, TCP sources with zero-rate profiles are starved completely (i.e., they do not get any bandwidth since all their packets are dropped by the queuing mechanism). This implies that with these configurations, neither WRED nor RIO can meet our first requirement for load-tolerance. For this simulation scenario and configuration of WRED and RIO, these mechanisms behave equally (i.e., the average throughput for TCP sources with rate profiles is the same for WRED and RIO).

Although the configuration of RIO used for the simulation presented in Figure 8 is the one recommended, we also evaluate a configuration for which RIO avoids starvation of *out* traffic (configuration 2 of RIO). Unfortunately, this configuration may give lower loss-rates to high precedence traffic than to low precedence traffic (i.e., the hierarchy among precedence levels may not be preserved). Figure 9 shows the results for RIO with this configuration together with the results for ltRIO.

RIO is, in this simulation, configured with max_th_in and min_th_in set to 100 packets, max_th_out to 200 packets, and min_th_out to 100 packets. Hence, the value of max_p_in is not relevant (since max_th_in and min_th_in are equal). The max_p_out parameter is set to 5 percent. ltRIO has the same configuration (i.e., the parameters present in both these mechanisms are set equally). The th_in parameter in ltRIO is set to 100 packets.

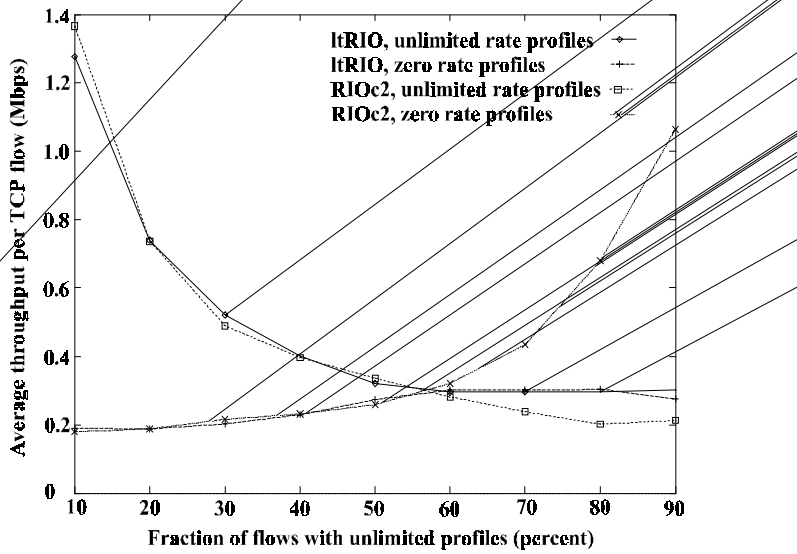

Figure 9 Throughput with ltRIO and RIO (configuration 2).

It can be seen in Figure 9 that ltRIO offers the same differentiation as RIO when the number of flows with unlimited rate profiles is less than 57 percent. Above that load, ltRIO behaves as RED while RIO no longer preserves the hierarchy between the *in* and *out* precedence levels. That is, for RIO, the loss-rates experienced by flows with unlimited rate profiles are higher than the loss-rates experienced by traffic with zero-rate profiles. For ltRIO, loss-rates are approximately equal for traffic with unlimited rate profiles and traffic with zero-rate profiles. Thus, ltRIO can offer sheltering without the risk of giving less quality to traffic tagged as *in* profile than to high precedence traffic.

[5] (500 kbps * 0,080 s) / 8000 bits = 5 packets on the fly on average.
[6] One segment of data is equal to the payload of one packet.

Figure 10 shows the results for WRT and WRED configured to offer relative differentiation (configuration 2 of WRED). WRED is in this simulation configured with max_th(0) and max_th(1) set to 200 packets, min_th(0) and min_th(1) to 100 packets, max_p(1) to 5 percent, and max_p(0) to 4 percent. As for the simulations presented in Figure 8, the parameters for the other six precedence levels are not relevant. The parameters present in both mechanisms are set to the same value. The max_th parameter in WRT is set to 100 packets.

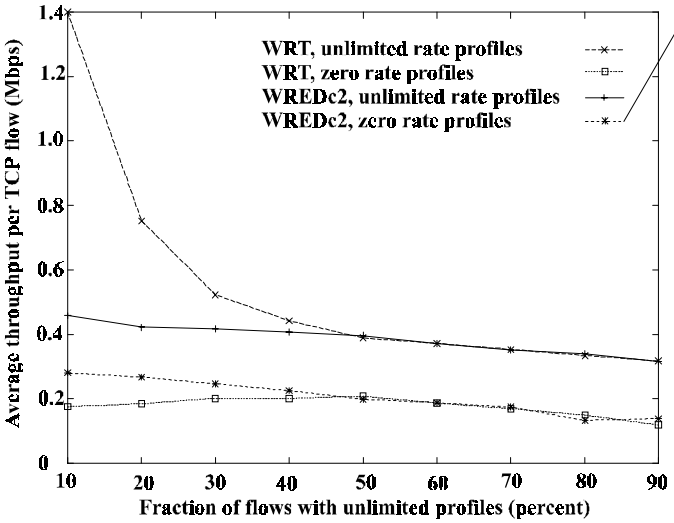

Figure 10 Throughput with WRT and WRED (configuration 2).

In Figure 10 it can be seen that both WRED and WRT offer a relative differentiation when the number of flows with unlimited rate profiles exceeds 50 percent. The average throughput per TCP flow experienced by flows with unlimited rate profiles is higher that the average throughput experienced by flows using zero-rate profiles. That is, for WRT, the loss-rates experienced by flows with unlimited rate profiles are lower than the loss-rates experienced by traffic with zero-rate profiles. This differentiation is the same as the one offered with WRED. When the number of flows with unlimited rate profiles is less than 50 percent, WRT offers the same differentiation as RIO and ltRIO.

Hence, with this particular configuration, WRT behaves as RIO and ltRIO if the number of flows with unlimited rate profiles are less than 50 percent and otherwise as WRED. This means that WRT preserves the hierarchy between drop precedence levels at all load scenarios tested. These observations indicate that WRT offers sheltering when low precedence traffic is properly controlled and relative differentiation otherwise.

5.3 Differentiation during Overload

Allowing sources to have unlimited rate profiles represents an extreme situation of overload since the admission control present is based only on the number of flows. We consider this as a worst case scenario when control of the aggregated traffic tagged as *in* profile has failed completely. Simulations with this kind of overload provide an indication of how sensitive the differentiation is to various amounts of TCP sources using unlimited rate profiles. To investigate this sensitivity, we apply a common profile of 5 Mbps to ten percent of all TCP flows (i.e., all flows from host S9). The sources with unlimited rate profiles vary between 0 and 80 percent. Hence, the graph in Figure 11 goes from 0 to 90 percent of flows with rate profiles instead of from 10 to 90 percent as in the previous graphs. This is to show the throughput the flows from S9 would have had if there was no overload (i.e., there are no TCP sources with unlimited rate profiles). In this simulation, we use the same configuration of ltRIO as for the simulation presented in Figure 9.

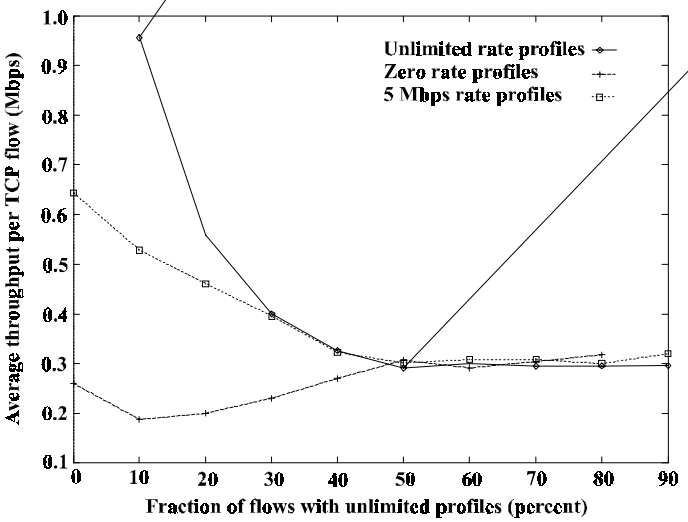

Figure 11 ltRIO under severe overload.

Figure 11 shows how the average throughput experienced by the ten controlled TCP sources degrades with an increasing number of flows using unlimited rate profiles. The controlled TCP sources are those with a common rate profile of 5 Mbps. It can be seen that a few uncontrolled TCP sources do not cause any severe degradation in throughput experienced by the TCP sources sharing the 5 Mbps profile. However, when the amount of uncontrolled sources goes above 45 percent, there is no differentiation whatsoever.

The rather gentle service degradation shown in Figure 11 presumes that the flows with unlimited rate profiles are responsive to network congestion signaling (i.e., packet drops). Unresponsive applications can cause this degradation to be much more drastic.

5.4 Bandwidth Allocation Limits

As mentioned above, ltRIO and WRT offer the same differentiation (i.e., sheltering) as RIO if avg_ql_in never exceeds th_in and the same differentiation (i.e., relative differentiation) as WRED otherwise. The question is then at which load differentiation changes from being sheltered to being relative. To study this issue, we have done a set of iterative simulations. Through these simulations, we find for two settings of ltRIO and three load scenarios (i.e., number of TCP flows with rate profiles) the maximum load of *in* traffic that can be supported without causing avg_ql_in to grow larger than th_in. The maximum load of *in* traffic is controlled by the aggregate TSW target rate. We denote the maximum aggregate TSW target rate as the *bandwidth allocation limit*.

Besides varying the target rate in the TSW rate estimator, we used the same configuration as in the simulation of ltRIO shown in Figure 9. Table 1 presents the maximum rates for two different settings of th_in, i.e., ½ and ¾ of max_th (the max_th parameter is set to 200 packets). For each of these settings, we have simulated three scenarios with 30, 40 and 50 percent of all flows having rate profiles.

th_in/max_th	½			¾		
Flows with rate profiles (%)	30	40	50	30	40	50
Maximum load (Mbps) (% of link speed)	14.04 46.8	13.13 43.8	14.35 47.8	22.13 73.8	24.11 80.4	22.13 73.8

Table 1 Bandwidth allocation limits with ltRIO.

Table 1 shows how the bandwidth allocation limit depends on how th_in is set in relation to max_th (Figure 5). For these simulations, setting th_in to ½ and ¾ of max_th_out result in a bandwidth allocation limit close to ½ and ¾ of the link speed respectively. Those relations cannot, however, be expected to hold for any configuration of ltRIO and any possible traffic load. This is because the bandwidth allocation limit will depend on the variation of avg_ql_in. The larger this variation is, the less bandwidth can be allocated without risking that avg_ql_in exceeds th_in. Such things as the window size in the TSW rate estimator, the configuration of the average queue length estimator in the AQM mechanism (RIO, ltRIO or WRT), and traffic burstiness affect the variation of avg_ql_in. Thus, the exact limit can only be found with real-time measurements. Nevertheless, our results indicate that a rough estimation of the bandwidth allocation limit for a certain link can be made based on the configuration of ltRIO.

5.5 Load-tolerance of RIO, ltRIO and WRT

In this section, we study the behavior of RIO, ltRIO and WRT when the load of *in* traffic (i.e., the amount of *bandwidth allocated*) is gradually increased to exceed the bandwidth allocation limit for one of these configurations (i.e., th_in is set to ½ of max_th). We examined the average throughput experienced by 10 TCP sources sharing a rate profile for *in* traffic of 5 Mbps in comparison with TCP sources having zero-rate profiles.

The total amount of bandwidth allocated is varied between 5 and 40 Mbps. Three sets of simulations were made with 30, 50 and 70 TCP sources using rate profiles. That is, 10 TCP sources share a rate profile of 5 Mbps and 20, 40 and 60 TCP sources have rate profiles between a total of 0 and 35 Mbps. The other 60, 40 and 20 TCP sources have zero-rate profiles.

Figure 12 shows the results for RIO configured to offer sheltering with the risk of starving high precedence traffic at overload (configuration 1 of RIO). This is the same configuration as the one used for the simulation presented in Figure 8.

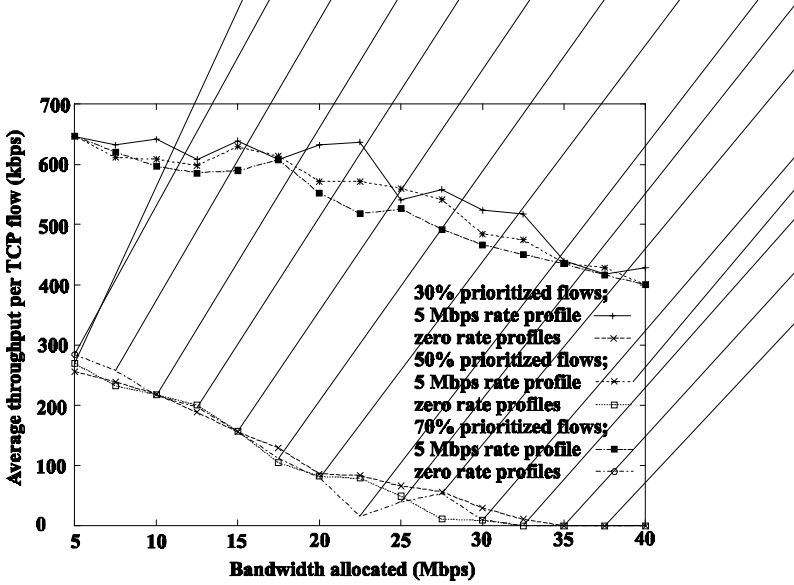

Figure 12 RIO under limited overload (configuration 1).

In Figure 12, it can be seen that the average throughput of the ten TCP sources with a common rate profile of 5 Mbps is more than 400 kbps[7] for all loads used in these simulations. No packet tagged as *in* profile were dropped when the amount of bandwidth allocated was equal or less than 30 Mbps. However, the TCP sources with zero-rate profiles experienced very high loss-rates. When the amount of bandwidth allocated was 35 Mbps or more, the TCP sources with zero-rate profiles did not get any data through. All *out* packets were then dropped. Hence, our first requirement for load-tolerance is not met. WRED behaves equally as RIO when configured as for the simulation presented in Figure 8 (configuration 1 of WRED).

Figure 13 shows the results for RIO configured to not guarantee low drop precedence traffic lower loss-rates (configuration 2 of RIO). This configuration is the same as in the simulation presented in Figure 9.

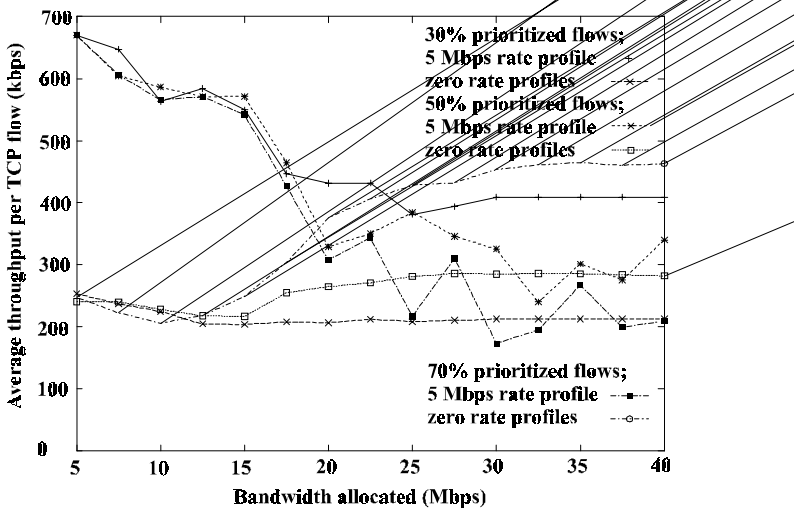

Figure 13 RIO under limited overload (configuration 2).

In Figure 13 it can be seen that, if the total amount of bandwidth allocated is less than 15 Mbps (i.e., ½ the link speed), the average throughput of the ten TCP sources with a common rate

[7] The expected average throughput of the ten TCP sources with a common rate profile of five Mbps is 500 kbps (5000 / 10 = 500).

profile of 5 Mbps is more than 500 kbps. When more bandwidth is allocated, the differentiation depends on the number of TCP sources having rate profiles. With 70 percent TCP sources having rate profiles, TCP sources with zero-rate profiles experience more throughput on average than the ten connections sharing a 5 Mbps rate profile (i.e., our second requirement for load-tolerance is not met). This behavior can also be observed in Figure 9.

In Figure 14, the results for ltRIO are shown. The same configuration of ltRIO as in the simulations presented in Figure 9 is used.

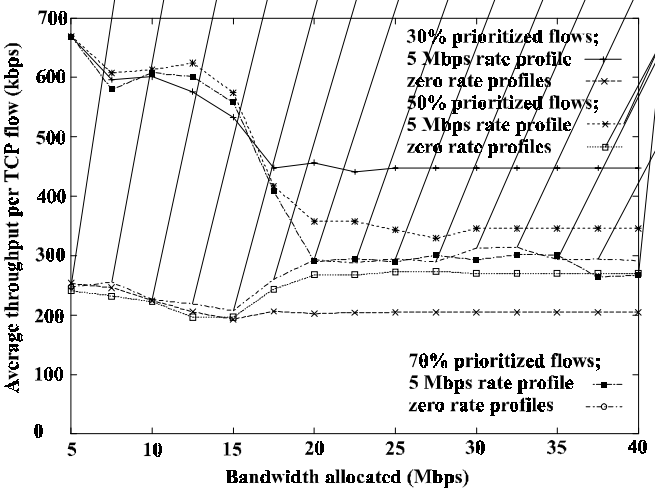

Figure 14 ltRIO under limited overload.

By comparing Figures 13 and 14 it can be seen that ltRIO offers a similar differentiation as RIO with configuration 2 if the total amount of bandwidth allocated is less than 15 Mbps. In addition, these two mechanisms offer a similar differentiation if the number of TCP sources having rate profiles is 30 percent. If there are more than 50 percent or more of the TCP sources with rate profiles, ltRIO behaves, as expected, as RED (i.e., no differentiation is offered).

Finally, in Figure 15 the results for WRT are shown. The same configuration of WRT as in the simulations presented in Figure 10 is used.

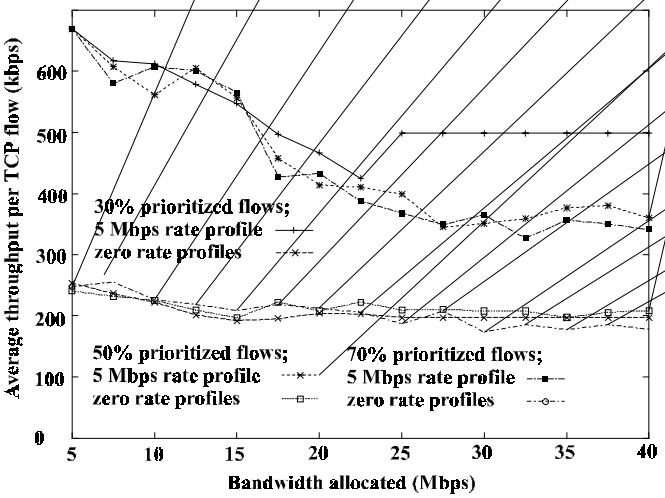

Figure 15 WRT under limited overload.

In Figure 15, it can be seen that WRT offers a similar differentiation as RIO and ltRIO if the total amount of bandwidth allocated is less than 15 Mbps. However, while RIO fails in preserving the hierarchy among precedence levels (configuration 2 of RIO, shown in Figure 9 and 13) and ltRIO behaves as RED at high loads (Figure 9 and 14), WRT provides relative differentiation when sheltering cannot be offered. This is also shown in Figure 10. RIO can be configured to preserve the hierarchy among precedence level, but *out* traffic may then suffer from starvation (configuration 1 of RIO, shown in Figure 8 and 12). Hence, WRT is the only queue mechanism of these three that can meet our requirements for load-tolerance when supporting conditional sheltering.

5.6 Summary of Simulation Results

We have shown that WRT offers, without reconfiguration, sheltering when *in* traffic is properly controlled and relative differentiation otherwise. RIO can be configured to offer sheltering, but cannot offer relative differentiation with any configuration. WRED can offer relative differentiation with one configuration and sheltering with another configuration. WRED cannot however offer both these kinds of differentiations without reconfiguration.

The sheltering WRT offers is shown to be the same as for RIO when the amount of *in* traffic is controlled below a certain limit and the relative differentiation is shown to be the same as for WRED.

Our simulations indicate that the bandwidth allocation limit when differentiation changes from sheltered to relative differentiation can roughly be estimated based on the configuration of WRT. However, the actual bandwidth that can be allocated will be less if avg_ql_in has a non-negligible variation.

6. Service Construction

In this paper, we introduce two properties, *sheltering* and *load-tolerance*, to characterize differentiating queuing mechanisms. We show that an AQM mechanism can be designed to support conditional sheltering and load-tolerance without reconfiguration.

A load-tolerant and conditional sheltering queuing mechanism is appealing for constructing predictable and advantageous end-to-end services. We say that a service is *advantageous* if it guarantees users equal or better service than users of the best-effort service[8]. If there is a high probability that users of a service receive an expected throughput, we say that the service is *predictable*.

Predictable and advantageous services can be constructed with a differentiating queuing mechanism that does not by itself prevent starvation. Then, to protect best-effort traffic against long-term starvation, one must however rely on accurate control of *in* traffic.

[8] A discussion on how to guarantee equal or better service than best-effort is provided in section 1.

Dynamic admission control is likely to be adequate in protecting best-effort traffic against long-term starvation. Statistically allocated service profiles may however not be adequate in protecting best-effort traffic against neither long-term nor transient starvation.

With a load-tolerant and conditional sheltering queuing mechanism, statistically allocated service profiles can be used for predictable and advantageous services without risking long-term and/or transient starvation of best-effort traffic. Moreover, transient congestion when using dynamic admission control can be avoided with such a mechanism. Using statistically allocated service profiles and dynamic admission control for predictable and advantageous services are discussed in section 6.1 and section 6.2 respectively.

6.1 Statistically Allocated Service Profiles

For statistically allocated, *destination-defined* (i.e., service profiles defining specific hosts or stub-networks as destinations), service profiles, periods of overload may occur if network routing topology changes. Moreover, for statistically allocated, *destination-independent* (i.e., service profiles defining specific networks as destinations), service profiles, periods of overload may be encountered for destinations that suddenly become more attractive than expected.

Since statistically allocated service profiles are likely to be *semi-statically* allocated, it may take some time to re-allocate these profiles after a failure has been detected. Hence, with statistically allocated service profiles, best-effort traffic cannot be protected against long-term and transient starvation unless the queuing mechanism is load-tolerant.

With a load-tolerant and conditional sheltering queuing mechanism, predictable and advantageous services can be constructed with statistically allocated service profiles without risking long-term or transient starvation. For such services, both destination-defined and destination-independent service profiles can be used. Rough estimations of *in* traffic load, which can be verified with periodic measurements, are likely to be sufficient for creating such services.

6.2 Dynamic Admission Control

With dynamic admission control, service requests are either accepted or rejected dependent on whether free capacity is currently available or not. The amount of free capacity can be estimated from measurements and/or by comparing the total service capacity and current service allocations.

When dynamic admission control is used, service profiles are often allocated dynamically as well. Dynamic admission control and allocation of service profiles can be made by a control system (e.g., a Bandwidth Broker [6] or a QoS agent [7][8]) or a signaling protocol (e.g., the Resource Reservation Protocol (RSVP) [3]).

For dynamically allocated, *destination-defined*, service profiles, the control system or the signaling protocol may fail in adapting fast enough to changes in network routing topology. Moreover, if measurement based admission control is used, the control mechanism may accidentally accept traffic causing temporary overload of *in* traffic until this condition is detected. *Destination-independent* service profiles might be possible to allocate and change dynamically if service preemption can be tolerated. Temporary overload of *in* traffic is however still likely to occur during the process of changing service profiles. Hence, although dynamic admission control is likely to protect best-effort traffic against long-term starvation, it may not protect this traffic against transient congestion. Some ISPs may accept transient starvation, but others may consider it important to avoid.

With a load-tolerant and conditional sheltering queuing mechanism, both transient and long-term starvation is avoided. Consequently, traffic control needs to be less conservative in considering rare network failures (which may cause changes in network routing topology). Less conservative, perhaps measurement-based, control mechanisms that dynamically allocate service profiles can increase the utilization of resources allocated for predictable and advantageous services. The utilization of these resources can be increased for both destination-defined and destination-independent service profiles.

7. Conclusions

In this paper we have evaluated the appropriateness of two AQM mechanisms, RIO [4] and WRED [5], in offering sheltering under different loads. A drop precedence level is said to be *sheltered* if traffic loads at higher precedence levels only have minor effects on the loss-rate experienced by traffic at this level. Sheltering is justified by requirements for service predictability. For this evaluation, we say that a differentiating queuing mechanism is *load-tolerant* if it can meet the following two requirements at overload:

- *Prevent starvation of high drop precedence traffic.*
  High drop precedence traffic must always get a useful share of the bandwidth available (i.e., even if low precedence traffic is not properly controlled).
- *Preserve hierarchy among drop precedence levels.*
  Traffic at a drop precedence level must always experience less drop probability than traffic at a higher drop precedence level.

Load-tolerance is appealing when using statistically allocated, destination-independent, service profiles. Longer periods of overload may then be encountered at topology changes or for destinations that suddenly become more attractive than expected. Without load-tolerance, long-term starvation of high precedence traffic may then occur. Avoiding long-term starvation of traffic at high drop precedence levels is desired when best-effort traffic is forwarded at such levels. Forwarding best-effort traffic with high drop precedence in the same queue as *in* traffic enables services guaranteeing equal or better treatment than best-effort.

We show, though simulations, that RIO and WRED cannot meet our requirements for load-tolerance when configured to offer sheltering. At overload, RIO can only meet one of the two requirements with one single configuration (i.e., with two different kinds of configurations, RIO can meet both require-

ments). WRED can only meet the requirements if configured to provide relative differentiation. Relative differentiation means, in this context, that traffic at a certain level experiences an average loss-rate defined in relation to the average loss-rate experienced by traffic at another level.

As neither RIO nor WRED can offer sheltering and meet our requirements for load-tolerance, we have proposed a new AQM mechanism, WRT. WRT calculates a separate average queue length for packets at a sheltered level. Packets at this level are not dropped as long as this average queue length stays below a certain threshold. If the average queue length for packets at sheltered level exceeds the threshold these packets are reclassified to a level that can be relatively differentiated to other levels. This action prevents high drop precedence traffic from being starved and preserves the hierarchy between precedence levels. WRT is thus load-tolerant according to our definition.

We have shown through simulations that WRT offers sheltering when *in* traffic is properly controlled and relative differentiation otherwise. Moreover, the sheltering offered by WRT is shown to be the same as for RIO when the amount of prioritized traffic is controlled below a certain limit and the relative differentiation is shown to be the same as for WRED.

8. References

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

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

[3] Braden R. et. al. (1997), *Resource Reservation Protocol (RSVP) - Version 1 Functional Specification*, IETF RFC2205, September 1997.

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

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

[6] Nichols K., Jacobson V. and Zhang L (1997), *A Two-bit Differentiated Services Architecture for the Internet*, November 1997, URL: ftp.ee.lbl.gov/papers /dsarch.pdf.

[7] Schelén O. and Pink S. (1998), *Resource Reservation Agents in the Internet*, The 9th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV98), July 1998.

[8] Schelén O. and Pink S. (1998), *Aggregating Resource Reservations over Multiple Routing Domains*, IWQoS'98, May 1998.

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

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

[11] UCB/LBNL/VINT Network Simulator - ns (version 2) (1999), URL: http://www-mash.cs.berkeley.edu/ns/.

[12] Heinanen J. et. al. (1999), *Assured Forwarding PHB Group*, IETF RFC 2597, June 1999.

[13] May M., Bolot J., Diot C. and Lyles B. (1999), *Reasons not to deploy RED*, IWQoS´99, June 1999.

[14] Jain R., Liu C., Goyal M. and Durresi A (2000), *Performance Analysis of Assured Forwarding*, IETF DRAFT February 2000.

[15] Elloumi O., De Cnodder S. and Pauwels K. (1999), *Usefullness of three drop precedences in Assured Forwarding service,* IETF DRAFT, July 1999.

[16] Seddigh N., Nandy B. and Pieda P. (1999), *Study of TCP and UDP Interaction for the AF PHB*, IETF DRAFT, September 1999.

[17] Kim H. (1999), *A fair Marker*, IETF DRAFT, April 1999.

[18] Lin D. and Morris R. (1997), *Dynamics of Random Early Detection*, SIGCOMM´97, September 1997

[19] Dovrolis C. and Ramanathan P. (1999), *A Case for Relative Differentiation Services and the Proportional Differentiation Model*, IEEE Network, 13(5): 26-34, September 1999 (special issue on Integrated and Differentiated Services in the Internet), URL: http://www.cae.wisc.edu/~dovrolis/homepage.html