Implementation of IP Proportional Differentiation with Waiting-Time Priority and Proportional Loss Rate dropper in Linux

Benjamin Gaidioz, Mathieu Goutelle, Pascale Primet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Implementation of IP Proportional Differentiation with Waiting-Time Priority and Proportional Loss Rate dropper in Linux

Benjamin Gaidioz (INRIA Reso, RESAM) Mathieu Goutelle (INRIA Reso, RESAM) and

Pascale Primet (INRIA Reso, RESAM)

N° 4511

Juillet 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RT--0266--FR+ENG

Implementation of IP Proportional Differentiation with Waiting-Time Priority and Proportional Loss Rate dropper in Linux

Benjamin Gaidioz (INRIA Reso, RESAM) Mathieu Goutelle (INRIA Reso, RESAM) and Pascale Primet (INRIA Reso, RESAM)

Thème 1 — Réseaux et systèmes
Projet Reso

Rapport de recherche n° 4511 — Juillet 2002 — 14 pages

Abstract: The proportional Diffserv (PDS) model is a model for providing service differentiation in IP. It ensures that a class of service obtains a performance (in terms of delay or loss rate) proportional to the performance obtained by an other class, according to a specific coefficient. The WTP and PLR schedulers provide proportional differentiation. They have not been widely tested today because commercials routers do not implement them. In this report we present an implementation in the Linux operating system of these schedulers and show experimental results we obtained with them.

Key-words: proportional differentiation, Waiting Time Priority scheduler, Proportional Loss Rate dropper, IP quality of service, Linux IP stack, Diffserv, service differentiation.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Mise en oeuvre de la différenciation proportionnelle IP dans Linux avec Waiting-Time Priority et Proportional Loss Rate dropper

Résumé : Le modèle de service Diffserv proportionnel (PDS) permet de fournir de la différenciation de service au niveau IP. Il assure qu'une classe de trafic donnée obtient une performance (en termes de délai de file d'attente ou de taux de perte) proportionnelle à celle qu'obtient une autre classe, ceci selon un coefficient déterminé. Les ordonnanceurs WTP et PLR permettent de pratiquer cette différenciation. Ils n'ont pas encore fait l'objet de nombreuses études car ils ne sont pas mis en oeuvre dans les routeurs commerciaux. Dans ce rapport, nous présentons leur mise en oeuvre dans le noyau Linux et montrons les résultats expérimentaux obtenus.

Mots-clés : différenciation proportionnelle, ordonnanceur Waiting Time Priority, ordonnanceur Proportional Loss Rate dropper, qualité de service IP, pile IP de Linux, Diffserv, différenciation de service.

1 Introduction

Quality of service (QoS) in IP is a well known challenge that has been widely studied. There is a need today to introduce some form of QoS at the IP level because of new applications with specific semantics differing from the classic file transfer. QoS is also expected to provide some privileged service levels to users that are willing to subscribe to "better than best-effort" services. The Diffserv [2] model is a scalable and robust solution to provide QoS at the IP level. In the model, an IP packet is associated to a specific class of traffic thanks to a class ID (among a relatively low number of IDs) that identifies a specific treatment (Per-Hop Behavior, PHB) experienced in each router along the path it is forwarded along. Service models have been separately defined. Standard models are today the Expedited Forwarding PHB [8, 3] and the Assured Forwarding PHB group [6]. They are both able to provide kind of absolute guarantees in delay or loss rate.

Beside these two models, several other models were proposed. Among them, we want to emphasize *relative differentiation models* like Scalable Share Differentiation [1], Paris Metro Pricing [12] and particularly proportional delay and loss rate differentiation [5]. Relative models have the particularity that they provide *relative* guarantees to classes, that is to say: a privileged class is guaranteed to obtain a relatively "better" performance than an other. Even if they provide a weaker service than absolute differentiation, they have the advantage of being much easier to deploy in network since they do not need admission control and a specific provisioning.

In [5], two specific schedulers (WTP, Waiting-Time Priority and PLR, Proportional Loss Rate dropper) are described. They permit to a router to practice a *proportional* delay or loss rate differentiation. Experiments that are conducted in the NS-2 [11] network simulator, show that both schedulers have a good accuracy.

In this report, we present an implementation of these schedulers as a unique Linux queuing discipline, as well as experimental results obtained with it. In sect. 2, we describe the schedulers and then show briefly how they were implemented in Linux in sect. 3. We present experimental results obtained on a Fast-Ethernet LAN in sect. 4. Sect. 5 is dedicated to related work and we give our conclusions and future work in sect. 6.

2 Description of the schedulers

The proportional differentiated services model presented in [5] is a case of *relative differentiated services*, which defines a specific order between classes without providing them absolute guarantees. It states that a metric for a class should be proportional to a parameter. The two metrics that are chosen are the delay and loss rate of the classes, which are agreed to be two characteristics of network quality. Each class is associated to two parameters: d_i for the delay differentiation and l_i for the loss rate differentiation.

The router should have a scheduler and queuing discipline able to handle the streams in such a manner that the relative ordering between classes are respected along the time. In [5] the authors describe a scheduler (*Waiting Time Priority*, WTP) and a queuing discipline (*Proportional Loss Rate dropper*, PLR) that achieve this differentiation.

2.1 Waiting Time Priority

The scheduler is illustrated on fig. 1 with four classes. Each class is associated to a specific queue. When a

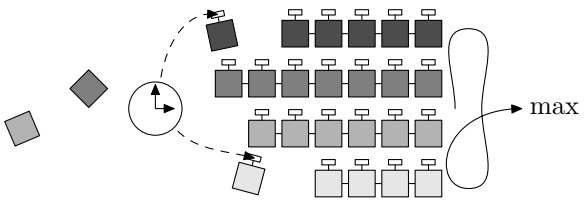

Figure 1: The WTP scheduler for proportional delay differentiation

packet is admitted in the router, the date is stored in it. This date permits to compute the waiting time δ of a packet when a packet is going to be emitted. A coefficient d_i is associated to queue i. When the router has to dequeue a packet, it computes the *normalized* waiting time $\bar{\delta}_i$ of each first packet of each queue by multiplying its real waiting-time δ_i by d_i. The router chooses to dequeue the packet with the highest *normalized* waiting time. Thus, the higher the coefficient d_i is, the quickest the class.

2.2 Proportional Loss Rate dropper

The queue management system is illustrated on fig. 2. The queuing discipline bases its differentiation on the average loss rate of all classes upon the *Loss History Table* (LHT), which gets the loss history of the last K received packets. On fig. 2, it is represented by the circular list. When a packet is received, an entry is added

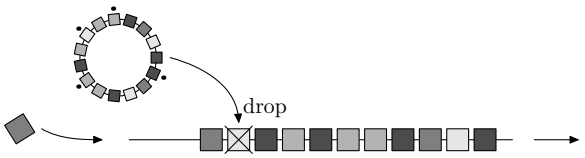

Figure 2: The PLR queuing discipline for proportional loss differentiation

in the LHT (by removing the oldest one) which contains the class of the packet (the color of the squares on the figure). Since the packet is in the queue, it is also marked in the LHT as *admitted*. Packet that are later dropped are marked as *dropped* (on the figure, packets which have a little black dot are marked as *dropped*). The average loss rates ρ_i are updated according to new packets admitted in the router and old packets removed from the LHT. When a congestion occurs, the router computes the *normalized* loss rates $\bar{\rho}_i$ by multiplying ρ_i by the l_i coefficient. The class that obtains the lowest $\bar{\rho}_i$ loses a packet while the new one is enqueued. The higher l_i is, the lower probability a class has of losing a packet.

3 Implementation

In this section, we present how WTP and PLR were implemented in Linux. Due to their complementarity, we have implemented them as a single Linux queuing discipline so that it is possible to practice at the same time both differentiations. The implementation of WTP and PLR is based upon the QoS support in the Linux kernel [4, 7]. Indeed, the scheme used by the Linux kernel to define queuing disciplines and schedulers makes it relatively straightforward to implement them. We have mainly to define three procedures: the **enqueue** procedure in charge of handling the new packets, the **drop** procedure, called when router capacity is reached and the **dequeue** procedure, called when a packet should be emitted.

3.1 The enqueue procedure

The history is treated as a cyclical queue, whose only the last packet is accessible with the command **history.end**. The command **history.next** is used to advance to the next position in the queue. First, when a packet is admitted in the router (see algorithm 1), the entrance time is stored in a field of the data structure of the packet. Then, the reception (**recv** array) and loss (**loss** array) statistics associated to the class of the packet which occupies the place at the end of the history are updated (it is going to be erased from the cyclical queue). Once the place is freed, the statistics are updated again according to the class of the new packet and a pointer to the position of the packet in the history is stored with the packet. Finally, the packet is added at the tail of the queue of its class. After the packet has been added, if the maximum capacity of the router has not been reached, the **pds_drop** procedure is called.

3.2 The dequeue procedure

When a packet has to be dequeued (see algorithm 2), the scheduler determines the class with the highest *normalized* delay. The **max_wait** variable contains the value of the highest delay. The **index_max** variable contains the index of class which has the maximum delay and is set initially to -1. In case of equality of the waiting times, the chosen class is the class with the lowest index (with the lowest priority). Finally, the chosen packet is extracted from the head of the queue and is returned as the result, except if all the queues are empty.

3.3 The drop procedure

In case of congestion, a packet must be dropped. The PDS model mentions that the queuing discipline must drop a packet from the class with the lowest *normalized* loss rate (see algorithm 3). We follow the same scheme as for the emission of a packet: each class is polled a packet to find the one which fits this criteria. In case of

Algo. 1 pds_enqueue function: admission of a packet.

```
    function pds_enqueue(paq: new packet): boolean
 2:
    paq.date_inq ← actual time
 4: { Delete a packet from the history and update the statistics. }
    if (history.end.status = DROP) then
 6:   loss[history.end.class]--
    end if
 8: recv[history.end.class]--

10: { Add the new packet at the end of the history. }
    history.end.class = paq.class
12: history.end.status = RECV
    paq.p_in_hist = ^history.end
14: recv[paq.class]++
    history.next { Advance to the next position. }
16:
    { Add the packet to the corresponding class. }
18: queues.add_tail(paq, paq.class)
    if (the capacity of the router is reached) then
20:   pds_drop() { see algorithm 3. }
      return(False)
22: end if
    return(True)
```

Algo. 2 pds_dequeue function: emission of a packet.

```
    function pds_dequeue(): boolean
 2: index_max: integer
    max_wait: time
 4: now ← actual time

 6: index_max ← −1
    for i from 1 to nb_class do { Poll all the classes. }
 8:   if (the class i isn't empty and
           d_i(now − queues(i).head.date_inq) > max_wait)
       then
         index_max ← i
10:      max_wait ← d_i(now − queues(i).head.date_inq)
      end if
12: end for

14: if there is really a packet to dequeue then
      return(files.dequeue_tail(index_max))
16: else
      return(NULL)
18: end if
```

Algo. 3 pds_drop function: drop policy.

```
    function pds_drop(): boolean
 2: index_min: integer
    rate_min: float
 4:
    index_min ← −1
 6: for i from 1 to nb_class do { Poll all the classes. }
      if (the class i isn't empty and p_i loss[i]/recv[i] < rate_min)
       then
 8:      index_min ← i
         rate_min ← p_i loss[i]/recv[i]
10:    end if
    end for
12: if there is really a packet to drop then
      { Dequeue the packet from the chosen class. }
14:   paq ← files.dequeue_tail(index_min)
      { Statistics must be updated. }
16:   paq.p_in_hist^.status ← DROP
      loss[index_min]++
18:   return(True)
    else
20:   return(False)
    end if
```

equality of the *normalized* loss rate, the class with the lowest index (with the lowest priority) is chosen. Once the class with the minimum loss rate is found, the last packet of its queue is dropped.

4 Validation

4.1 Experimental testbed

For the validation, a cluster of four Athlon™ machines (850 MHz, 128 Mo RAM) running Debian GNU/Linux 3.0 (2.4.18 kernel) with 100baseTX Full Duplex connections was used (see figure 3). Two machines run as traffic

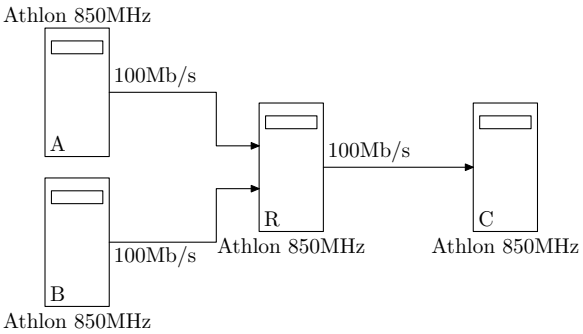

Figure 3: The platform used for the experimental study.

sources (A and B), one host as destination (C), while the fourth (R) was configured as router. The MGEN Toolset [10] was used as traffic generator. Loss rate is measured by host C according to log files of MGEN. Queuing delay is harder to measure because on the opposite to loss rate in our testbed, local differentiation through one router does not lead to the same end-to-end differentiation. In our experiments, the queuing delay experienced by a packet is measured in the router R between the call to *enqueue* and the call to *dequeue*. There may be some extra delay added in the router itself before enqueuing or after dequeuing but it is not our

goal to implement the delay differentiation scheduler so that it takes into account as best as possible the delay experienced in the router.

The router was configured to handle 8 classes with a limit of 80 packets the history depth set to $K = 1500$ packets. During the experiment, the two sources inject traffic into the router in order to obtain a congestion: the A source sends UDP packets inside classes 1, 3, 5 and 7 while B sends UDP packets inside 2, 4, 6 and 8 classes, both at the bit-rate of 1100 packets per second.

To get an idea of the reactivity of the scheduler, the bit-rate of the senders is not constant during the whole experiment : at time $t = 1.5\,\mathrm{s}$, B decreases progressively the bit-rate of class 8 from 1100 to 400 packets/s and increases it again progressively to reach again a bit-rate of 1100 packets/s at $t = 2\,\mathrm{s}$. A has the same behavior at $t = 2.5\,\mathrm{s}$ and $t = 3.5\,\mathrm{s}$ respectively for class 1 and class 5. At $t = 4\,\mathrm{s}$, A and B increase progressively the bit-rate of all classes from 1100 to 1500 packets/s up to $t = 9\,\mathrm{s}$.

We hereby present two types of results: the first one is the evolution of the two metrics (delay elapsed in the router or loss rate experienced by classes of traffic) during the experiment. The time spent in the router is measured in micro-seconds and the loss rate in percentage while the time in abscissa is shown in seconds since the start of the experiment. The second is the evolution of the ratios between the performances of the classes and class 1, to see if the router respects the coefficients set in the scheduler.

In subsect. 4.4 and 4.5, we check the performance of one differentiation only (coefficients are different one to each other) while the other differentiation is practiced with all coefficients set to 1. Setting all coefficients to 1 is not equivalent to practicing no differentiation at all. It is a first step to see if both differentiations can be practiced at the same time with no side effects. Then in subsect. 4.6 we see if the performance still has the same accuracy while both differentiations are practiced at the same time. Subsect. 4.7 is dedicated to special cases of differentiation where we show some "side effects" of the schedulers.

4.2 Coefficient setting

In all experiments, differentiation is sometimes practiced in delay, loss rate or both. Each time a specific differentiation is practiced, the coefficient setting is the same. The ideal setting we wanted to have is that a class i has a coefficient of $1/i$. The class with the highest coefficient (class 1) is the one that obtains the best performance (say the lowest delay). Class 2 with coefficient $1/2$ then obtains a delay that is twice delay of the first class, class 3 obtains a delay that is three times delay of class 1, etc. This setting gives a differentiation where performances are regularly spaced. Due to integer rounding, the settings were relatively equivalent to the ideal setting. Coefficients are 157, 78, 52, 39, 31, 26, 22 and 19. The proportional performances we expect with this setting are:

$$\frac{157}{19} = 8.26, \quad \frac{157}{22} = 7.13, \quad \frac{157}{26} = 6.03, \quad \frac{157}{31} = 5.06,$$
$$\frac{157}{39} = 4.02, \quad \frac{157}{52} = 3.01, \quad \frac{157}{78} = 2.01, \quad \frac{157}{157} = 1$$

instead of 8, 7, 6, 5, 4, 3, 2 and 1.

4.3 Driver configuration

Delay differentiation with large ratios can appear to be affected by the architecture of the operating system itself. In the kernel, the driver we used (`3c59x`) was configured by default to dequeue (calls to **dequeue**) 16 packets in a row and pass the address pointers to the NIC that sends them one by one. Even if the packets are emitted later sequentially by the NIC (and received sequentially also), 16 packets are logged as sent at the same time while in fact being received sequentially so that delay ratios computed upon calls to **dequeue** are wrong. We had to patch the driver (`3c59x`) to get better accuracy of the scheduler by reducing the number of packets simultaneously passed by the driver to the NIC from 16 to 2. This modification seems not to affect the performance of the scheduling in our case. However, we do not claim that this setting would have no impact on any machine. We simply noticed we could use this to obtain more accurate measurements that would reflect well if the scheduler itself is efficient.

Results that are shown are measured with the new driver.

4.4 Validation of delay differentiation with equal loss rate differentiation

In this section, we run delay differentiation only and show how well the scheduler performs it. Loss rate coefficient setting is $l_i = 1$ (that is to say, equal loss rate). Delay coefficient setting is $19, \ldots, 157$. Results are shown on fig. 4. Leftmost figure shows the delays while rightmost figure shows the ratios.

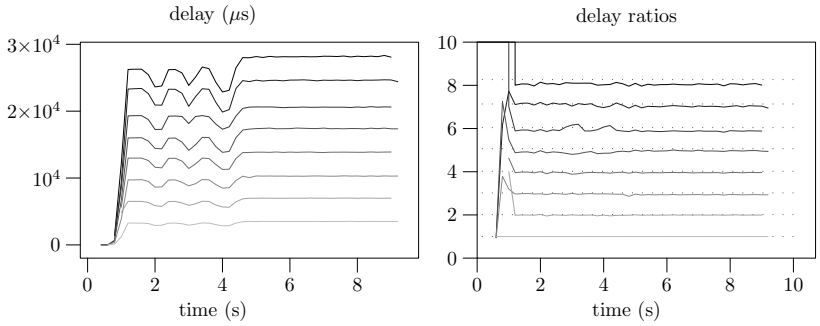

Figure 4: Evolution of queuing delays (left). Evolution of the ratios of queuing delays (right).

The expected differentiation is obtained. The differentiation starts only when a sufficient number of flows send traffic (about 0.7 s after the beginning). Peaks in the delay graph correspond to dates where a class modifies its rate. During these peaks of delay, the ratios are well ensured. We emphasize the fact that accidents on the graphs are magnified because the delay fluctuations of a class are multiplied by the associated d_i factors. The graph shows that the delay differentiation is accurate.

4.5 Validation of loss rate differentiation with equal delay differentiation

In this section, we observe the performance of loss rate differentiation. We measure an average loss rate over 0.2 s. Coefficients setting for classes is here opposite to the differentiation differentiation set in 4.2. That is to say, class 1 gets the highest loss rate since it was the quickest in the previous experiment. On figure 5, we show the evolution of the loss rates and loss rate ratios. The differentiation only occurs when there is a congestion,

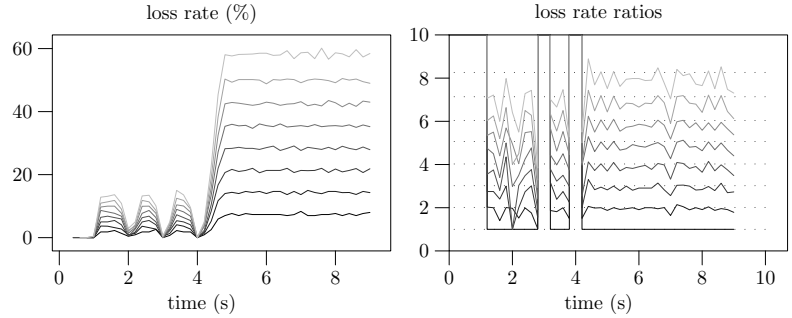

Figure 5: Evolution of the loss rates and loss rate ratios.

i.e. when the router capacity is reached. During the first part of the experiment, each time a class decreases its bit-rate, the overall loss rate becomes low (close to zero). During variations of bit-rates, the differentiation is not very accurate because the loss ratios are low. There are some places where the ratios go to a very large value when the lowest loss rate (the one the others are compared to) is very close to zero. The second part of the experiment gives better results because load is high.

4.6 Validation of both simultaneous delay and loss differentiation

In this experiment, we set the coefficients like in the previous experiments but in an asymmetric manner. The quickest class obtain the highest loss. Results are shown on fig. 6. Both differentiations are pretty well ensured.

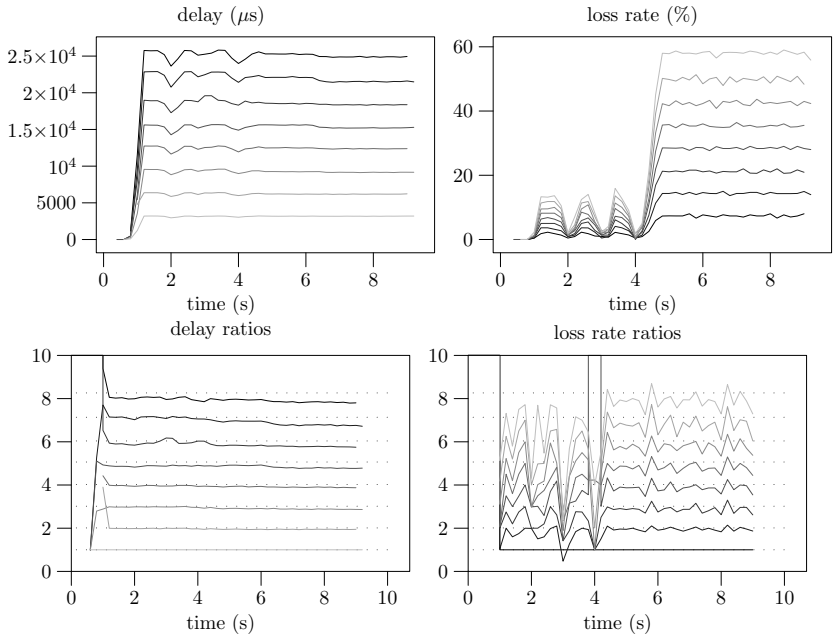

Figure 6: Evolution of performance in delay and loss rate while both differentiations are practiced at the same time.

4.7 Side effects

In this section, we emphasize some "side effects" of the differentiation in particular cases. In the first part we show that a lower load may increase the delay of all classes. Then we show that delay differentiation can have an effect on the accuracy of loss rate differentiation. Then, we show how loss rate differentiation is affected by the history size of PLR.

4.7.1 Impact of load and delay differentiation on delay

Usually, a lower load leads to a lower delay. This has been seen in the experiment described in 4.4 where delay decreases when a class decreases its rate. In this section, we show how the opposite can happen when the differentiation is large. In this experiment, the slowest class is the only one that modifies its rate by decreasing it and then increasing it again. Results are shown on fig. 7. With a soft differentiation (the one described

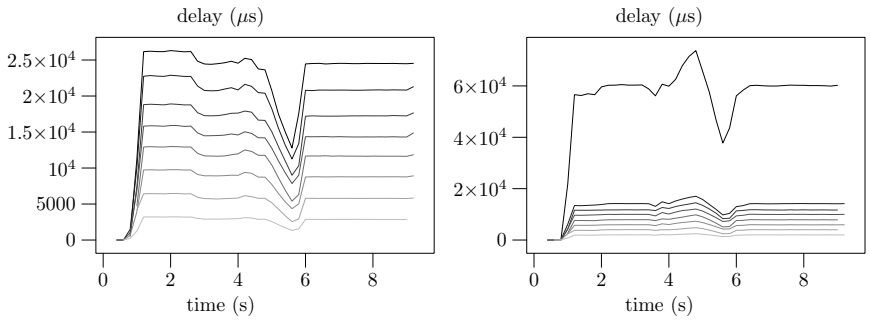

Figure 7: With a large delay differentiation, a lower load can lead to a higher delay. On the right figure, the delay increases when load decreases.

in 4.2), when the slowest class decreases its rate, it leads to a lower delay. Then the delay increases again when the rate is increased (see leftmost graph of fig. 7). The second graph (rightmost) was obtained with a larger delay differentiation for the slowest class (the coefficient is 5 instead of 19, the class is approximately four times slower). When it decreased its rate, it first results in a increase of the delay and then on a decrease. This is due to the fact that since the class is the slowest, its queue is very large compared to others (since its rate and

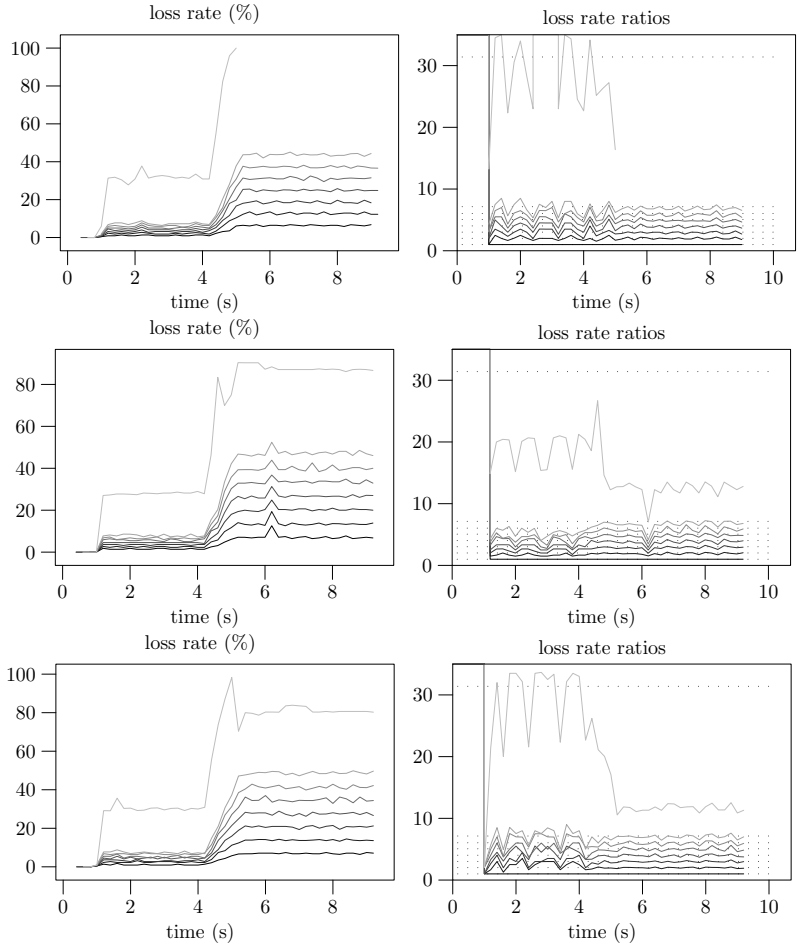

Figure 8: A large loss rate differentiation can be affected by delay differentiation. With delay differentiation, the class which should obtain at least 100% loss, obtains less than 100% loss.

loss rate coefficients are the same). The router is filled with a large number of its packets, so that the others classes have a short queue. When its rate decreases, the router is still full because the load is too large. But less packets of the slowest class are in the router so that the other classes get larger queues. This results in delays increasing. Then, the load is lower than what the router can handle so that the delay decreases.

This side effect is due to the fact that the router capacity is measured as the sum of all the queue length.

4.7.2 Impact of delay differentiation of loss rate differentiation

In this section, we show that when the load is so large that the class with the highest loss rate is supposed to lose all its packets, if the delay differentiation makes it the quickest class, it can affect the loss rate differentiation. In this section, the differentiation is practiced with the class with the highest loss rate having a coefficient of 5 instead of 19 so that its loss rate is expected to be much higher than in the previous experiments. Fig. 8 shows the results obtained by practicing no delay differentiation, then the usual delay differentiation, then a large delay differentiation (the same with 5 instead of 19).

While loss rate is theorically supposed to be higher than 100%, a 100% loss rate is only ensured when no delay differentiation is practiced. Loss rate ratio is seriously wrong with the quickest class because it is less than 100%. What happens is that when delay differentiation is practiced, the queue of the highest loss rate class is very short and has a good priority. Since CBR flows are not completely CBR flows (due to operating system delay in some places in source machines for example), there are some very short periods of time where the router is not full so that packets are forwarded (loss rate is zero). During these periods, the loss rate ratios are not compared and packets are enqueued anyway. If the queue of the high loss rate class is short and has priority (delay differentiation is practiced), it is possible that packets get forwarded during these periods, and as soon as it gets full again, it drops all packets of the queue while accepting others (since the average loss rate is

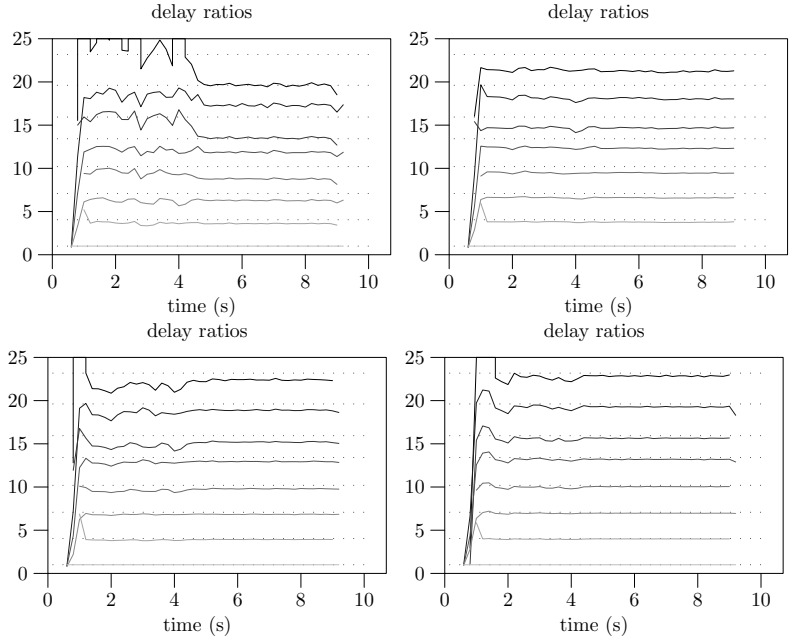

Figure 9: Impact of the queue length on delay differentiation (from left to right, then from top to bottom, lengths of 40, 80, 160 and 320 packets).

lower than 100%, all packets are dropped to make it become sufficiently high). When no delay differentiation is practiced, the queue has no priority. The router accepts packets during short periods of low load but the WTP does not dequeue them soon because they are new in the router and are not given a higher priority. Congestion occurs again quickly and packets are dropped them before they get forwarded. Thus, the queue length of the high loss rate class is low (because usually, all of its packets are dropped) but packets wait in the queue because of WTP and get dropped later.

4.7.3 Impact of the queue length on a large delay differentiation

In this experiment, we show how a large delay differentiation can be affected by queue length. Coefficients are 11, 13, 16, 19, 25, 36, 63 and 255. The proportional performances we expect with this setting are:

$$\frac{255}{11} = 23.18, \quad \frac{255}{13} = 19.61, \quad \frac{255}{16} = 15.93, \quad \frac{255}{19} = 13.42,$$
$$\frac{255}{25} = 10.20, \quad \frac{255}{36} = 7.08, \quad \frac{255}{63} = 4.04, \quad \frac{255}{255} = 1$$

These coefficients are close to 1, 4, 7, 10, 13, 16, 19 and 23. Evolution of ratios are shown on fig. 9 where the experiment is conducted with queue capacities of 40, 80, 160 and 320. It is obvious that the longer the queue, the more accurate the differentiation in delay. This is probably due to experiment oscillations (flows are not very CBR, the operating system is not real-time) that are more visible with short delays. However, a large queue capacity leads to unacceptable large queuing delays. On a faster network with a faster machine, the capacity could be higher without leading to large delays. In this case, the accuracy should be fine and queuing delays low.

4.7.4 Impact of history size on loss rate differentiation

We check here what impact the history size can have on the accuracy of loss rate differentiation. Previous experiments were run with an history size of 1500. We present on fig. 10 and 11, results obtained with history sizes of 100, 400, 800, 1500, 5000 and 10000. The values shown on the graphs are average values computed on a period of 0.2 s, that is to say around 1600 packets when the load is high. Graphs show that an history size of 100 is too short to permit the router to practice a good differentiation. The value of 100 is probably too small compared to the rate oscillations of sources due to the operating system scheduling in source machines and router. Loss rate differentiation is more constant with 400 and 800 but the ratios are also wrong. Then, with

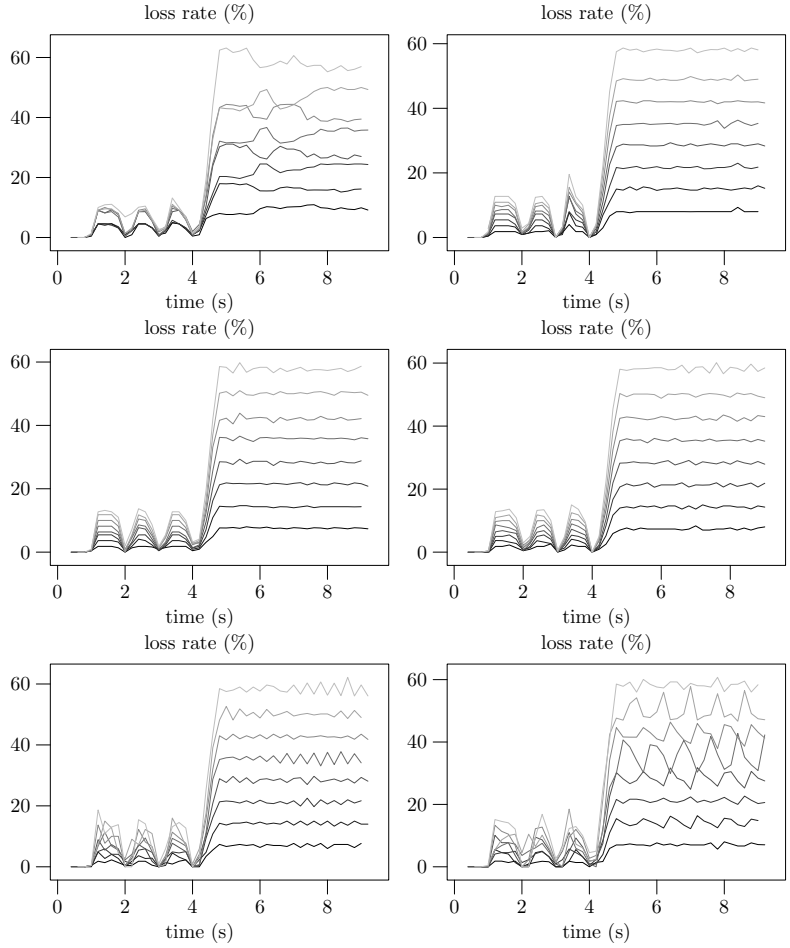

Figure 10: Evolution of loss rates with history sizes of (from left to right then top to bottom) 100, 400, 800, 1500, 5000 and 10000.

1500, the ratios are pretty fine. With history sizes of 5000 and 10,000, ratios are good in the second part of the experiment even if they oscillate a lot. However, the history sizes are too large to make the differentiation fine in the first part of the experiment where it is obvious that the three little peaks are not very well differentiated.

5 Related Work

As far as we know, there is no implementation in Linux of these schedulers that was released to the public. In [9], the author shows that WTP gives less accurate results in a relatively low load because it does not practice an average delay differentiation. The author also gives a way to tune the coefficient according to the expected differentiation and the load. In our study, we did not check the average delay with a coarse grain. However, our implementation could be used to experimentally check the adaptation of coefficients proposed in [9].

6 Conclusion

In this article we present an implementation of the Waiting-Time Priority and Proportional Loss Rate dropper schedulers as a single Linux kernel queuing discipline. Experiments first show that both delay differentiation and loss rate differentiation have a good accuracy. When practiced simultaneously, there is no obvious reason that would make them work less efficiently since they do not interact heavily. However, we show and demonstrate how loss rate differentiation accuracy can be significantly affected when delay differentiation is practiced at the same time. We also show that the size of the PLR scheduler history table has an impact on the accuracy of the loss rate differentiation so that it is a critical parameter. We think there is no obvious solution to these

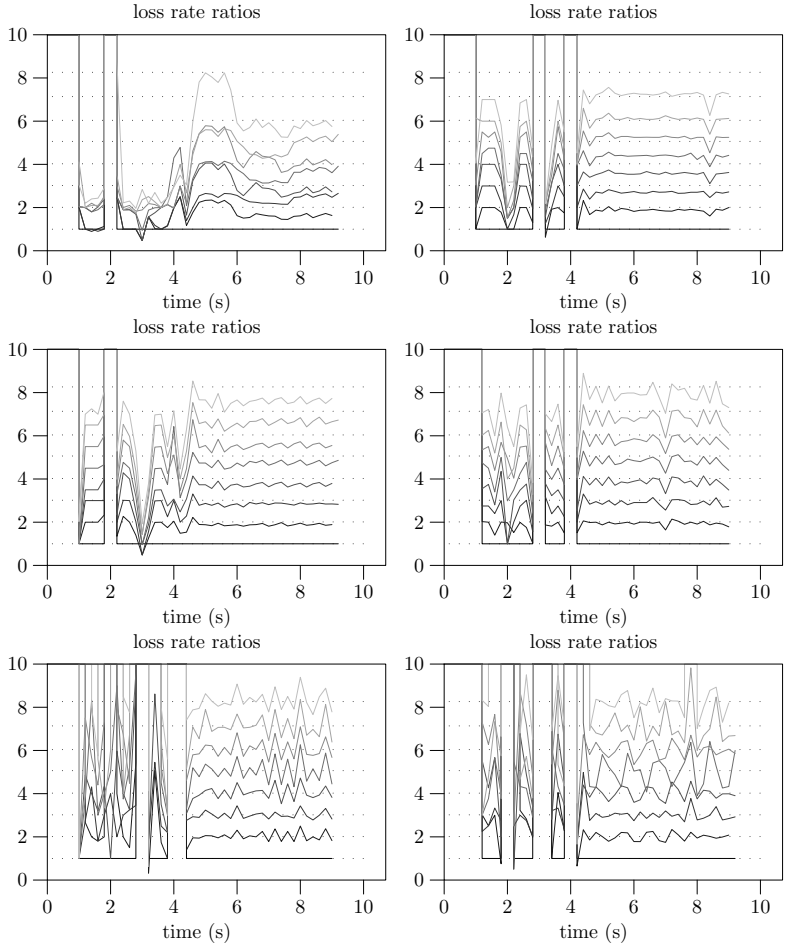

Figure 11: Evolution of loss rates *ratios* with history sizes of (from left to right then top to bottom) 100, 400, 800, 1500, 5000 and 10000.

issues because they are inherent to PLR. We are thinking that an implementation of a proportional loss rate differentiation based on early drops (a RED-like system) would be more efficient than PLR.

References

[1] Albert Banchs and Robert Denda. A scalable share differentiation architecture for elastic and real-time traffic. In *International Workshop on Quality of Service (IWQoS) '00*, pages 42–51, Pittsburgh, PA USA, June 2000. Institute of Electrical and Electronics Engineers (IEEE)/IFIP.

[2] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Services. Internet Request For Comments RFC 2475, Internet Engineering Task Force (IETF), December 1998.

[3] Bruce Davie, Anna Charny, Jon C. R. Bennett, Kent Benson, Jean-Yves Le Boudec, William Courtney, Shahram Davari, Victor Firoiu, and Dimitrios Stiliadis. An expedited forwarding phb (per-hop behavior). Internet Request For Comments RFC 3246, Internet Engineering Task Force (IETF), March 2002.

[4] Differentiated Services on Linux. `http://diffserv.sourceforge.net/`.

[5] Constantinos Dovrolis and Parameswaran Ramanathan. A case for relative differentiated services and the proportional differentiation model. *IEEE Network*, September 1999.

[6] Juha Heinanen, Fred Baker, Walter Weiss, and John Wroclawski. Assured Forwarding PHB Group. Internet Request For Comments RFC 2597, Internet Engineering Task Force (IETF), June 1999.

[7] Bert Hubert, Gregory Maxwell, Remco Van Mook, Martijn Van Oosterhout, Paul B. Schroeder, and Jasper Spaans. Linux advanced routing & traffic control. HOWTO, March 2002.

[8] Van Jacobson, Kathleen Nichols, and Kedar Poduri. An expedited forwarding phb. Internet Request For Comments RFC 2598, Internet Engineering Task Force (IETF), June 1999.

[9] John C.S. Lui Matthew K.H. Leung. Characterization and performance evaluation for the proportional delay differentiated services. In *Proceedings of ICNP 2000*, Osaka, Japan, November 2000.

[10] Mgen toolset web site. `http://manimac.itd.nrl.navy.mil/MGEN/`.

[11] Network simulator ns-2 web site. `http://www.isi.edu/nsnam/ns/`.

[12] Andrew Odlyzko. Paris metro pricing: The minimalist differentiated services solution. In *International Workshop on Quality of Service (IWQoS) '99*, pages 159–161, London, England, June 1999. Institute of Electrical and Electronics Engineers (IEEE)/IFIP.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399