Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

Baker
Finzi
TTTech Computertechnik AG
Frances
ISAE-SUPAERO
Kuhn
CNES
Lochin
Mifdaoui
ISAE-SUPAERO

October 22, 2018

# Priority Switching Scheduler
# draft-finzi-priority-switching-scheduler-04

## Abstract

We detail the implementation of a network rate scheduler based on both a packet-based implementation of the generalized processor sharing (GPS) and a strict priority policies. This credit based scheduler called Priority Switching Scheduler (PSS), inherits from the standard Strict Priority Scheduler (SP) but dynamically changes the priority of one or several queues. Usual scheduling architectures often combine rate schedulers with SP to implement DiffServ service classes. Furthermore, usual implementations of rate scheduler schemes (such as WRR, DRR, ...) do not allow to efficiently guarantee the capacity dedicated to both AF and DF DiffServ classes as they mostly provide soft bounds. This means excessive margin is used to ensure the capacity requested and this impacts the number of additional users that could be accepted in the network. PSS allows a more predictable output rate per traffic class and is a one fit all scheme allowing to enable both SP and rate scheduling policies within a single algorithm.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction . . . . . 2
  1.1. Context and Motivation . . . . . 2
  1.2. Definitions and Acronyms . . . . . 3
  1.3. Priority Switching Scheduler in a nutshell . . . . . 3
2. Priority Switching Scheduler . . . . . 5
  2.1. Specification . . . . . 5
  2.2. Implementation with three traffic classes and one controlled queue . . . . . 9
  2.3. Implementation with n controlled queues . . . . . 10
3. Usecase: benefit of using PSS in a Diffserv core network . . 12
  3.1. Motivation . . . . . 12
  3.2. New service offered . . . . . 14
4. Security Considerations . . . . . 14
5. Acknowledgements . . . . . 15
6. References . . . . . 15
  6.1. Normative References . . . . . 15
  6.2. Informative References . . . . . 15
Authors' Addresses . . . . . 16

## 1. Introduction

### 1.1. Context and Motivation

To enable DiffServ traffic classes and share the capacity offered by a link, many schedulers have been developed such as Strict Priority, Weighted Fair Queuing, Weighted Round Robin or Deficit Round Robin.

In the context of a core network router architecture aiming at managing various kind of traffic classes, scheduling architectures require to combine a Strict Priority (to handle real-time traffic) and a rate scheduler (WFQ, WRR, ... to handle non-real time traffic) as proposed in [RFC5865]. For all these solutions, the output rate of a given queue often depends on the amount of traffic managed by other queues. PSS aims at reducing the uncertainty of the output rate of selected queues, we call them in the following controlled queues. Additionally, compared to previous cited schemes, the scheduling scheme proposed is simpler to implement as PSS allows to both enable Strict Priority and Fair Queuing services; is more flexible following the wide possibilities offered by this setting; and does not require a virtual clock as for instance, WFQ.

## 1.2. Definitions and Acronyms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

- AF: Assured Forwarding;
- BLS: Burst Limiting Shaper;
- DRR: Deficit Round Robin
- DF: Default Forwarding;
- EF: Expedited Forwarding;
- PSS: Priority Switching Scheduler;
- QoS: Quality-of-Service;
- FQ: Fair Queuing
- SP: Strict Priority
- WFQ: Weighted Fair Queuing
- WRR: Weighted Round Robin

## 1.3. Priority Switching Scheduler in a nutshell

```
                                                 ____________________
                                                | p_low[i]  p_high[i] |
                                         ------|____________________|
                              sets()    |               ^
                               _________|__             |
   PSS controlled             |         |  |            | selects()
       queue i ------------>|  p[i]=  v  |            |
                              |            |        credit[i]
                      .       |      .     |            ^
                      .       |      .     |            | updates()
                      .       |      .     |            |
   non-active                 |            |------------------> output
   PSS queue j ------------>|    p[j]    |                    traffic
                              |            |
                      .       |      .     |
                      .       |      .     |
                      .       |      .     |
                              |____________|
                            Priority Scheduler
```

Figure 1: PSS in a nutshell

As illustrated in Figure 1, the principle of PSS is based on the use of credit counters (detailed in the following) to change the priority of one or several queues. Each controlled queue i is characterized by a current priority state p[i], which can takes two priority values: {p_high[i], p_low[i]} where p_high[i] the highest priority value and p_low[i] the lowest. This idea follows a proposal made by the TSN Task group named Burst Limiting Shaper [BLS]. For each controlled queue i, each current priority p[i] changes between p_low[i] and p_high[i] depending on the associated credit counter credit[i]. Then a Priority Scheduler is used for the dequeuing process, i.e., among the queues with available traffic, the first packet of the queue with the highest priority is dequeued.

The main idea is that changing the priorities adds fairness to the Priority Scheduler. Depending on the credit counter parameters, the amount of capacity available to a controlled queue is bounded between a minimum and a maximum value. Consequently, good parameterization is very important to prevent starvation of lower priority queues.

The service obtained for the controlled queue with the switching priority is more predictable and corresponds to the minimum between a desired capacity and the residual capacity left by higher priorities. The impact of the input traffic sporadicity from higher classes is thus transfered to non-active PSS queues with a lower priority.

Finally, PSS offers much flexibility as both controlled queues with a guaranteed capacity (when two priorities are set) and queues scheduled with a simple Priority Scheduler (when only one priority is set) can conjointly be enabled.

## 2. Priority Switching Scheduler

### 2.1. Specification

For the sake of clarity and to ease the understanding of the PSS algorithm, we consider the case where only one queue is a controlled queue. This corresponds to three traffic classes EF, AF and DF where AF is the controlled queue as shown in Figure Figure 2.

```
                queues    priority   ___
               ________             |   \
       EF--->|________|-----{1}----+    \
                                   |     \
               ________            |      \
       AF--->|________|-----{2,4}--+  PSS  --->
                                   |      /
               ________            |     /
       DF--->|________|-----{3}----+    /
                                   |___/
```

Figure 2: PSS with three traffic classes

As previously explained, the PSS algorithm defines for the controlled queue a low priority denoted p_low, and a high priority denoted p_high associated to a credit counter denoted credit, which manages the priority switching. Considering Figure 2, the priority p[AF] of the controlled queue AF will be switched between two priorities where p_high[AF] = 2 and p_low[AF] = 4. The generalisation of PSS algorithm to n controlled queues is given in Section 2.3.

Then, each credit counter is defined by:

- a minimum level: 0;

- a maximum level: LM;

- a resume level: LR such as 0 <= LR < LR;

- a reserved capacity: BW;

- an idle slope: I_idle = C * BW, where C is the link output capacity;

o a sending slope: I_send = C - I_idle;

The available capacity (denoted C) is mostly impacted by the guaranteed capacity BW. Hence, BW should be set to the desired capacity plus a margin taking into account the additional packet due to non-preemption as explained below:

the value of LM can negatively impact on the guaranteed available capacity. The maximum level determines the size of the maximum sending windows, i.e, the maximum uninterrupted transmission time of the controlled queue packets before a priority switching. The impact of the non-preemption is as a function of the value of LM. The smaller the LM, the larger the impact of the non-preemption is. For example, if the number of packets varies between 4 and 5, the variation of the output traffic is around 25% (i.e. going from 4 to 5 corresponds to a 25% increase). If the number of packets sent varies between 50 and 51, the variation of the output traffic is around 2%.

The credit allows to keep track of the packet transmissions. However, keeping track the transmission raises an issue in two cases: when the credit is saturated at LM or at 0. In both cases, packets are transmitted without gained or consumed credit. Nevertheless, the resume level can be used to decrease the times when the credit is saturated at 0. If the resume level LR is 0, then as soon as the credit reaches 0, the priority is switched and the credit saturates at 0 due to the non-preemption of the current packet. On the contrary, if LR > 0, then during the transmission of the non-preempted packet, the credit keeps on decreasing before reaching 0 as illustrated in Figure 3.

Hence, the proposed value for LR is Lmax * BW, with Lmax the maximum packet size of the controlled queue. With this value, there is no credit saturation at 0 due to non-preemption.

A similar parameter setting is described in [Globecom17], to transform WRR parameter into PSS parameters, also in the case of a three classes DiffServ architecture.

The priority change depends on the credit counter as follows:

o initially, the credit counter starts at 0;

o the change of priority p[i] of controlled queue i occurs in two cases:

  * if p[i] is currently set to p_high[i] and credit[i] reaches LM;

  * if p[i] is currently set to p_low[i] and credit[i] reaches LR;

- o when a packet of the controlled queue is transmitted, the credit increases (is consumed) with a rate I_send, else the credit decreases (is gained) with a rate I_idle;

- o when the credit reaches LM, it remains at this level until the end of the transmission of the current packet (if any);

- o when the credit reaches LR and the transmission of the current packet is finished, in the abscence of new packets to transmit in the controlled queue, it keeps decreasing at the rate I_idle until it reaches 0. Finally, the credit remains to 0 until the start of the transmission of a new packet.

Figure 3 and Figure 4 give two examples of credit and priority changes of a given queue. First, Figure 3 gives an example when the controlled queue sends its traffic continuously until the priority changes (this traffic is represented with @ below the x-axis of this figure). Then, the credit reaches LM and the last packet is transmitted although the priority have changed. Other traffic is thus sent (represented by o) uninterruptedly until the priority changes back. Figure 4 illustrates a more complex behaviour. First, this figure shows when a packet with a priority higher than p_high[i] is available, this packet is sent before the traffic of queue i. Secondly, when no traffic with a priority lower than p_low[i] is available, then traffic of queue i can be sent. This highlights the non-blocking nature of PSS and that p[i] = p_high[i] (resp. p[i] = p_low[i]) does not necessarily mean that traffic of queue i is being sent (resp. not being sent).

```
        ^ credit
        |         |                   |
        | p_high  |       p_low       |  p_high
    LM  |- - - - -++++++- - - - - - - |- - - -+++
        |        +|    |+             |      +
        |I_send + |    |  +  I_idle   |     +
        |      +  |    |    +         |    +
        |     +   |    |      +       |   +
        |    +    |    |        +     |  +
        |   +     |    |          +   | +
    LR  |  +      |    |            + |+
    0   |-+- - - -|- - |- - - - - - - +- - - - - >
                      |              |          time
        @@@@@@@@@@@@@@@@oooooooooooooo@@@@@@@@@@

        @ controlled queue traffic
        o other traffic
```

Figure 3: First example of queue credit evolution and priority switching.

```
        ^ credit
        |                         |
        |         p_high          |     p_low
    LM  + - - - - - - - - - - - -++++ - - - - - - -+
        |                        +|  |+           +
        |               ++     + |  |  +       +
        |              + | +  +  |  |    +   +
        |     ++      +  |   +   |  |      +
        |    +|  +   +   |   |   |  |      |
        |   + |     +    |   |   |  |      |
    LR  +--+--|-----|----|---|---|--|------|-------
    0   +-+- -| - - |- - |- -|- -|- |- - - |- - - - >
              |     |    |   |      |      |        time
        @@@@@@oooooo@@@@@oooo@@@@@@@@oooooo@@@@@@@

        @ controlled queue traffic
        o other traffic
```

Figure 4: Second example of queue credit evolution and priority switching.

Finally, for the dequeuing process, a Priority Scheduler selects the appropriate packet using the current priority values.  In other words, among the queues with packets enqueued, the first packet of the queue with the highest priority is dequeued (usual principle of SP).

## 2.2. Implementation with three traffic classes and one controlled queue

The new dequeuing algorithm is presented in the PSS Algorithm in Figure 5 and consists in a modification of the standard SP. The credit of the controlled queue and the dequeuing timer denoted timerDQ are initialized to zero. The initial priority is set to the highest value p_high. First, we compute the difference between the current time and the time stored in timerDQ (line #3). The duration dtime represents the time elapsed since the last credit update, during which no packet from the controlled queue was sent, we call this the idle time. Then, if dtime > 0, the credit is updated by removing the credit gained during the idle time that just occurred (lines #4 and #5). Next, timerDQ is set to the current time to keep track of the last time the credit was updated (line #6). If the credit reaches LR, the priority changes to its high value (lines #7 and #8). Then, with the updated priorities, SP algorithm performs as usual: each queue is checked for dequeuing, highest priority first (lines #12 and #13). When the queue selected is the controlled queue, the credit expected to be consumed is added to the credit variable (line #16). The time taken for the packet to be dequeued is added to the variable timerDQ (line #17) so the transmission time of the packet will not be taken into account in the idle time dtime (line #3). If the credit reaches LM, the priority changes to its low value (lines #18 and #19). Finally, the packet is dequeued (line #22).

```
Inputs: credit, timerDQ, C, LM, LR, BW, p_high, p_low
currentTime = getCurrentTime()
dtime = currentTime - timerDQ
   if dtime > 0 then:
      credit = max(credit - dtime * C * BW, 0)
      timerDQ = currentTime
      if credit < LR and p = p_low then:
         p = p_high
      end if
   end if
end for
for each priority level, highest first do:
   if length(queue[i]) > 0 then:
      if queue[i] is the controlled queue then:
         credit =
           min(LM, credit + size(head(queue[i])) * (1 - BW))
         timerDQ = currentTime + size(head(queue[i]))/C
         if credit >= LM and p = p_high then:
            p = p_low
         end if
      end if
      dequeue(head(queue[i]))
      break
   end if
end for
```

Figure 5: PSS algorithm

PSS algorithm implements the following functions:

- o getCurrentTime() uses a timer to return the current time;
- o length(q) returns the length of the queue q;
- o head(q) returns the first packet of queue q;
- o size(f) returns the size of packet f;
- o dequeue(f) activates the dequeing event of packet f.

### 2.3. Implementation with n controlled queues

The algorithm can be updated to support n controlled queues. In this context, the credits of each queue i must be stored in the table creditList[i]. Each controlled queue i has its own dequeuing timer stored in the table timerDQList[i]. Likewise for each controlled queue, LM[i], LR[i], BW[i], p_low[i] and p_high[i] are respectively stored in LMList[i], LRList[i], BWList[i], p_lowList[i] and

p_highList[i]. A controlled queue i is characterized by p_lowList[i] > p_highList[i] (as priority 0 is the highest priority for SP). The current priority of a controlled queue is stored in p[i]. Each controlled queue must have distinct priorities.

As an example, Figure Figure 6 extends Figure 2 to n controlled queues.

```
                     queues      prio     ___
                   ________              |   \
    Admitted EF--->|________|-----{1}----+    \
                                         |     \
                   ________              |      \
  Unadmitted EF--->|________|-----{2}----+       \
                                         |        \
                   ________              |         \
             AF1-->|________|-----{3,6}--+  PSS     --->
                                         |         /
                   ________              |        /
             AF2-->|________|-----{4,7}--+       /
                                         |      /
                   ________              |     /
             DF--->|________|-----{5}----+    /
                                         |___/
```

Figure 6: PSS with three traffic classes

```
Inputs: creditList[], timerDQList[], C, LMList[], LRList[],
        BWList[],p_highList[], p_lowList[]
 1   for each queue i with p_highList[i] < p_lowList[i] do:
 2      currentTime = getCurrentTime()
 3      dtime = currentTime - timerDQList[i]
 4      if dtime >0 then:
 5         creditList[i] =
             max(creditList[i] - dtime * C * BWList[i], 0)
 6         timerDQList[i] = currentTime
 7         if credit[i] < LRList[i] and p[i] = p_lowList[i] then:
 8            p[i] = p_highList[i]
 9         end if
10      end if
11   end for
12   for each priority level pl, highest first do:
13      if length(queue(pl)) > 0 then:
14         i = queue(pl)
15         if p_highList[i] < p_lowList[i] then:
16            creditList[i] =
                min(LMList[i],
                creditList[i] + size(head(i)) * (1 - BWList[i]))
17            timerDQList[i] = currentTime + size(head(i))/C
18            if creditList[i] >= LMList[i]
                 and p[i] = p_highList[i] then:
19               p[i] = p_lowList[i]
20            end if
21         end if
22         dequeue(head(i))
23         break
24      end if
25   end for
```

Figure 7: PSS algorithm

The general PSS algorithm also implements the following function:

o  queue(pl) returns the queue i associated to priority pl.

## 3. Usecase: benefit of using PSS in a Diffserv core network

### 3.1. Motivation

The DiffServ architecture defined in [RFC4594] and [RFC2475] proposes a scalable mean to deliver IP quality of service (QoS) based on handling traffic aggregates. This architecture follows the philosophy that complexity should be delegated to the network edges while simple functionalities should be located in the core network.

Thus, core devices only perform differentiated aggregate treatments based on the marking set by edge devices.

Keeping aside policing mechanisms that might enable edge devices in this architecture, a DiffServ stateless core network is often used to differentiate time-constrained UDP traffic (e.g. VoIP or VoD) and TCP bulk data transfer from all the remaining best-effort (BE) traffic called default traffic (DF). The Expedited Forwarding (EF) class is used to carry UDP traffic coming from time-constrained applications (VoIP, Command/Control, ...); the Assured Forwarding (AF) class deals with elastic traffic as defined in [RFC4594] (data transfer, updating process, ...) while all other remaining traffic is classified inside the default (DF) best-effort class.

The first and best service is provided to EF as the priority scheduler attributes the highest priority to this class. The second service is called assured service and is built on top of the AF class where elastic traffic such as TCP traffic, is intended to achieve a minimum level of throughput. Usually, the minimum assured throughput is given according to a negotiated profile with the client. The throughput increases as long as there are available resources and decreases when congestion occurs. As a matter of fact, a simple priority scheduler is insufficient to implement the AF service. TCP traffic increases until reaching the capacity of the bottleneck due to its opportunistic nature of fetching the full remaining capacity. In particular, this behaviour could lead to starve the DF class.

To prevent a starvation and ensure to both DF and AF a minimum service rate, the router architecture proposed in [RFC5865] uses a rate scheduler between AF and DF classes to share the residual capacity left by the EF class. Nevertheless, one drawback of using a rate scheduler is the high impact of EF traffic on AF and DF. Indeed, the residual capacity shared by AF and DF classes is directly impacted by the EF traffic variation. As a consequence, the AF and DF class services are difficult to predict in terms of available capacity and latency. To overcome these limitations and make AF service more predictable, we propose here to use the newly defined Priority Switching Scheduler (PSS).

Figure 8 shows an example of the Data Plane Priority core network router presented in [RFC5865] modified with a PSS. The EF queues have the highest priorities to offer the best service to real-time traffic. The priority changes set the AF priorities either higher (3,4) or lower (6,7) than CS0 (5), leading to capacity sharing (CS0 refers to Class Selector codepoints 0 and is usually refered to DF as explained in [RFC7657]). Another example with only 3 queues is described in [Globecom17]. Thank to the increase predictability, for the same minimum guaranteed rate, the PSS reserves a lower percentage

of the capacity than a rate scheduler. This leaves more remaining capacity that can be guaranteed to other users.

```
                      prio              ___
                                       |   \
 Admitted EF------{p[AEF] = 1}--------+    \
                                       |     \
                                       |      \
 Unadmitted EF----{p[UEF] = 2}--------+       \
                                       |        \
                                       |         \
 AF1--{p_high[AF1]=3, p_low[AF1]= 6}--+  PSS     --->
                                       |         /
                                       |        /
 AF2--{p_high[AF2]=4, p_low[AF2]= 7}--+       /
                                       |      /
                                       |     /
 CS0------------{p[CS0] = 5}----------+    /
                                       |___/
```

Figure 8: PSS applied to Data Plane Priority (we borrow the syntax from RCF5865)

## 3.2. New service offered

The new service we seek to obtain is:

- o for EF, the full capacity of the output link;
- o for AF the minimum between a desired capacity and the residual capacity left by EF;
- o for DF (CS0), the residual capacity left by EF and AF.

As a result, the AF class has a more predictable available capacity, while the unpredictability is reported on the DF class. With good parametrization, both classes also have a minimum rate ensured. Parameterization and simulations results concerning the use of a similar scheme for core network scheduling are available in [Globecom17]

# 4. Security Considerations

There are no specific security exposure with PSS that would extend those inherent in default FIFO queuing or in static priority scheduling systems. However, following the DiffServ usecase proposed in this memo and in particular the illustration of the integration of PSS as a possible implementation of the architecture proposed in

[RFC5865], most of the security considerations from [RFC5865] and more generally from the differentiated services architecture described in [RFC2475] still hold.

## 5. Acknowledgements

This document was the result of collaboration and discussion among a large number of people. In particular the authors wish to thank David Black, Ruediger Geib, Vincent Roca for reviewing this draft and Victor Perrier for the TUN/TAP implementation of PSS. At last but not least, a very special thanks to Fred Baker for his help.

## 6. References

### 6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

### 6.2. Informative References

[BLS] Gotz, F-J., "Traffic Shaper for Control Data Traffic (CDT)", IEEE 802 AVB Meeting , 2012.

[Globecom17] Finzi, A., Lochin, E., Mifdaoui, A., and F. Frances, "Improving RFC5865 Core Network Scheduling with a Burst Limiting Shaper", Globecom , 2017, <http://oatao.univ-toulouse.fr/18448/>.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <https://www.rfc-editor.org/info/rfc2475>.

[RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006, <https://www.rfc-editor.org/info/rfc4594>.

[RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <https://www.rfc-editor.org/info/rfc5865>.

[RFC7657] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <https://www.rfc-editor.org/info/rfc7657>.

## Authors' Addresses

Fred Baker
Santa Barbara, California 93117
USA

Email: FredBaker.IETF@gmail.com

Anais Finzi
TTTech Computertechnik AG
Schoenbrunner Strasse 7
Vienna 1040
Austria

Phone: 0043158534340
Email: anais.finzi@tttech.com

Fabrice Frances
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse 31400
France

Email: fabrice.frances@isae-supaero.fr

Nicolas Kuhn
CNES
18 Avenue Edouard Belin
Toulouse 31400
France

Email: nicolas.kuhn@cnes.fr

Emmanuel Lochin
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse  31400
France

Phone: 0033561338485
Email: emmanuel.lochin@isae-supaero.fr

Ahlem Mifdaoui
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse  31400
France

Email: ahlem.mifdaoui@isae-supaero.fr