Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: December 1, 2018

Baker

Finzi
Frances
Lochin
Mifdaoui
ISAE-SUPAERO
May 30, 2018

Priority Switching Scheduler

draft-finzi-priority-switching-scheduler-02

Abstract

We detail the implementation of a network scheduler that aims at isolating time constrained and elastic traffic flows from best-effort traffic. This scheduler inherits from the priority scheduler (PS) but dynamically changes the priority of one or several queues. Usual implementations of rate scheduler schemes (such as WRR, DRR, ...) do not allow to efficiently guarantee the capacity dedicated to both AF and BE 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. To cope with this issue, this memo presents a credit based scheduler mechanism called Priority Switching Scheduler (PSS) that allows a more predictable output rate per traffic class.

Status of This Memo

Copyright Notice

described in the Simplified BSD License.

Table of Contents

1. Introduction

1.1. Context and Motivation

To share the capacity offered by a link, many fair schedulers have been developed, such as Weighted Fair Queuing, Weighted Round Robin or Deficit Round Robin. However, with these well-known solutions, the output rate of a given queue depends on the amount of traffic crossing other queues. Our proposal 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, this solution is simpler to implement mainly because it does not require a virtual clock, and more flexible thanks to the wide possibilities offered by the setting of different priorities.

1.2. Requirements Language

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.

1.3. Priority Switching Scheduler in a nutshell

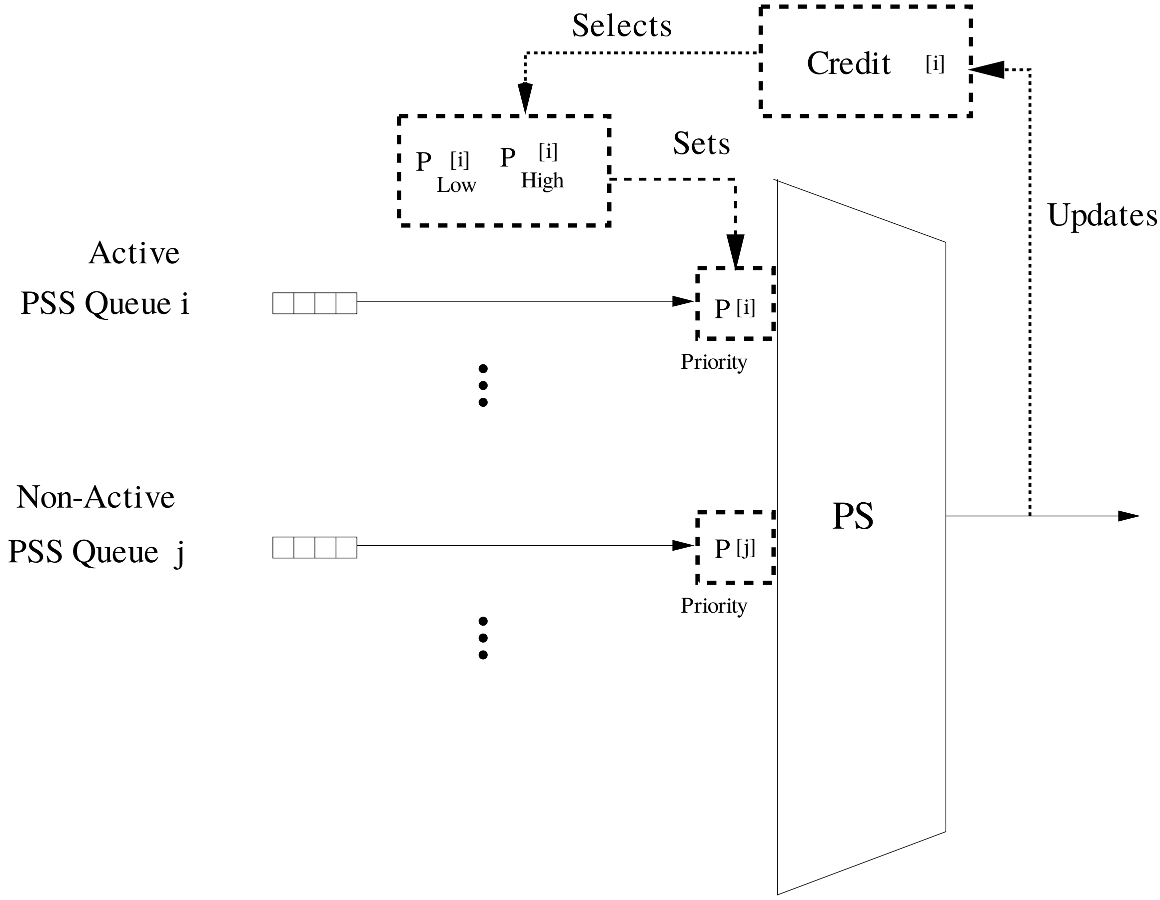

Figure 1: PSS in a nutshell

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. The idea follows a proposal made by the TSN Task group named Burst Limiting Shaper. For each controlled queue i, each priority denoted p[i], changes between two values denoted p_low[i] and p_high[i], depending on the associated credit counter, i.e., credit[i]. Then a Priority Scheduler is used for the dequeuing process, e.g., 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 i) controlled queues with a guaranteed capacity (when two priorities are set), ii) 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

The PSS algorithm defines for each queue q a low priority, p_low[q], and a high priority, p_high[q]. Each PSS controlled queue q with p_high[q] < p_low[q] is associated to a credit counter credit[q] which manages the priority switching. Each credit counter is defined by:

- a minimum level: 0;
- a maximum level: LMs[q];
- a resume level: LRs[q]
- a reserved capacity: BWs[q]
- an idle slope: Iidle[q] = C * BWs[q];
- a sending slope: Isend[q] = C - Iidle[q];

The available capacity is mostly impacted by the guaranteed capacity BWs[q]. Hence BWs[q] should be set to the desired capacity plus a margin taking into account the additional packet due to non-premption as explained below:

the value of LMs[q] 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-premption is as a function of the value of LMs[q]. The smaller the LMs[q], the larger the impact of the non-premption 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, there are two cases keeping track of the transmission raises an issue: when the credit is saturated at LMs[q] 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 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 LRs[q]>0, then during the transmission of the non-preempted packet, the credit keeps on decreasing before reaching 0 as illustrated in Figure 2.

Hence, the proposed value for LRs[q] is LRs[q]= Lmax(MC(q))*BWs[q], with MC(q) the queues such as k in MC(q) -> p_low[q] > (p_low[k] or p_high[k])> p_high[q], and Lmax(qs) the maximum size of the queues qs. With this value, there is no credit saturation at 0 due to non-preemption.

Finally, we propose to use the following parameters of a controlled queue q:

- BWs[q]= desired_BWs[q] + 1/(N-1)
- LMs[q]= (N-1) * Lmax(q) * (1 - BWs[q])
- LRs[q]= Lmax(MC(q)) * BWs[q]

with N the maximum number of packet of queue q set uninterrupted (taking into account the non-preemption) and desired_BWs[q] the percentage of desired available capacity.

A similar parameter setting is described in [Globecom17], to transform WRR parameter into PSS parameters, in the specific case of 3-classes DiffServ architecture.

The priority change depends on the credit counter as follows:

- initially, the credit counter starts at 0;
- the change of priority p[q] of queue q occurs in two cases:
    - if p[q] = p_high[q] and the credit reaches LMs[q];
    - if p[q] = p_low[q] and credit reaches LRs[q];

- when a packet of queue q is transmitted, the credit increases with a rate Isend[q], else the credit decreases with a rate Iidle[q];
- when the credit reaches LMs[q], it remains at this level until the end of the transmission of the current packet;
- when the credit reaches 0, it remains at this level until the start of the transmission of a queue q packet.

Figure 2 and Figure 3 show two examples of credit and priority changes of a given queue q. First, in Figure 2, we show an example when the controlled queue q sends its traffic continuously until the priority change. Then other traffic is also sent uninterruptedly until the priority changes back. In Figure 3, we propose a more complex behaviour. First, this figure shows when a packet with a priority higher than p_high[q] is available, this packet is sent before the traffic of class q. Secondly, when no traffic with a priority lower than p_low[q] is available, then traffic of queue q can be sent. This highlight the non-blocking nature of PSS and that p[q] = p_high[q] (resp. p[q] = p_low[q]) does not necessarily mean that traffic of queue q is being sent (resp. not being sent).

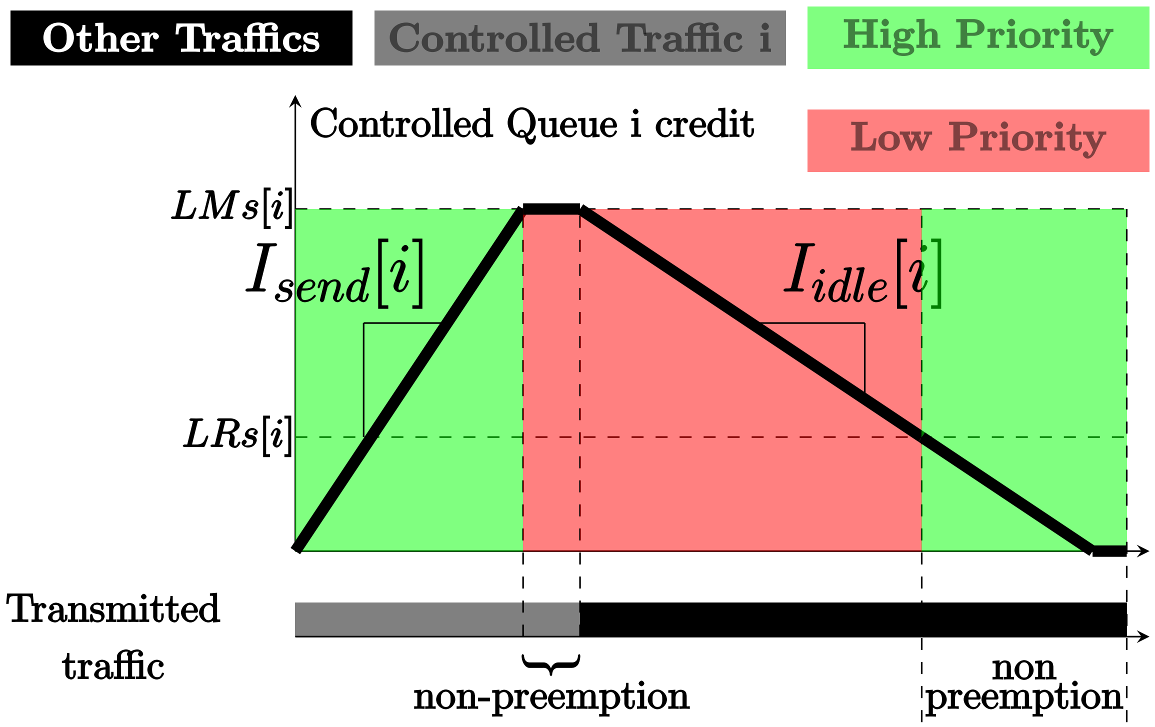

Figure 2: First example of queue q credit and priority behaviors

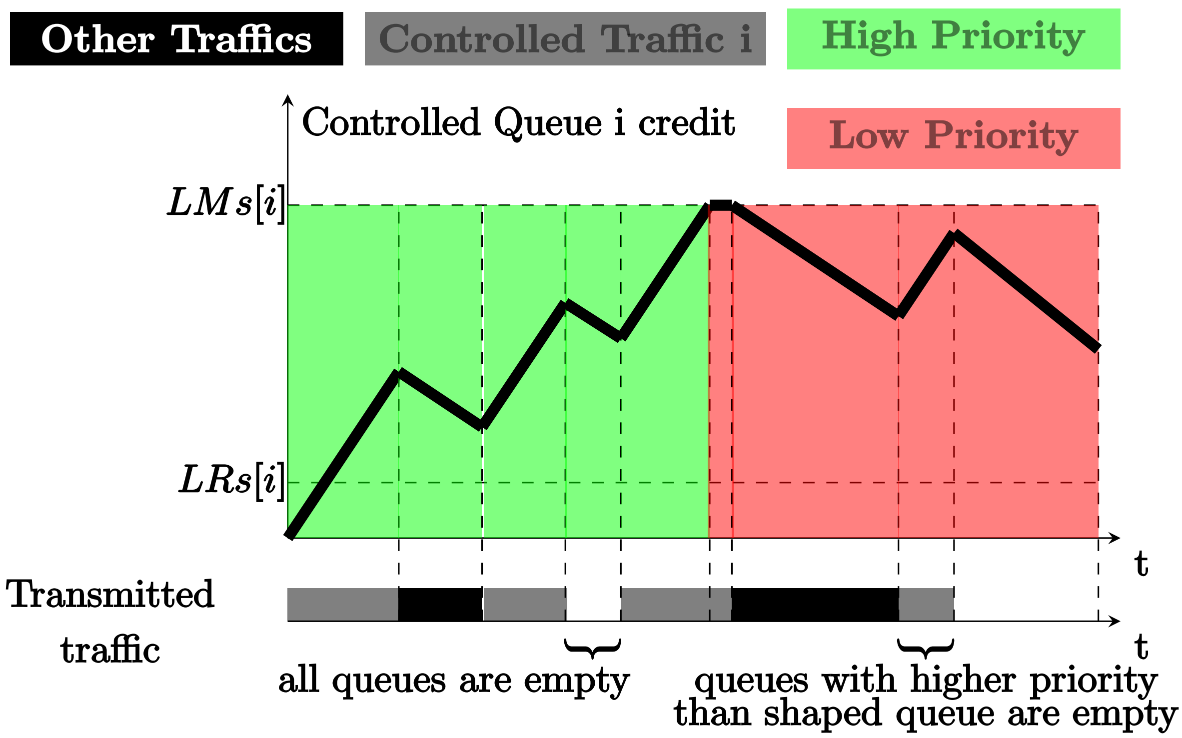

Figure 3: Second example of queue q credit and priority behaviors

Finally, for the dequeuing process, a Priority Scheduler selects the appropriate packet using the current p[q] values, e.g., among the queues with available traffic, the first packet of the queue with the highest priority is dequeued.

2.2. Implementation

The new dequeuing algorithm is presented in the PSS Algorithm. The credit of each queue q, denoted credit[q], and the dequeuing timer denoted timerDQ[q] are initialized to zero. The initial priority is set to the high value p_high[q]. First, for each queue with p_high[q] > p_low[q], the difference between the current time and the time stored in timerDQ[q], is computed (lines 2 and 3). The duration dtime[q] represents the time elapsed since the last credit update, during which no packet of the controlled queue q was sent, we call this the idle time. Then, if dtime[q]>0, the credit is updated by removing the credit gained during the idle time that just occurred (lines 4 and 5). Next, timerDQ[q] is set to the current time to keep track of the time the credit is last updated (line 6). If the credit reaches LRs[q], the priority changes to its high value (lines 7 and 8). Then, with the updated priorities, the priority scheduler performs as usual: each queue is checked for dequeuing, highest priority first (lines 12 and 13). When a queue q is selected with p_high[q] < p_low[q], the credit expected to be consumed is added to credit[q] variable (line 16). The time taken for the packet to be dequeued is added to the variable timerDQ[q] (lines 13 and 14) so the transmission time of the packet will not be taken into account in the idle time dtime[q] (line 2). If the credit reaches LMs[q], the priority changes to its low value (lines 18 and 19). Finally, the packet is dequeued (line 22).

```
Inputs: credits, timerDQs, C, LMs,LRs,BWs,p_highs, p_lows
 1   currentTime = getCurrentTime()
 2   for each queue q with p_high[q] < p_low[q] do:
 3      dtime[q] = currentTime-timerDQ[q]
 4      if dtime[q]>0 then:
 5         credit[q] = max(credit[q]-dtime[q].C.BWs[q],0)
 6         dtime[q] = currentTime
```

```
      if credit[q]<LRs[q] and p[q] = p_low[q] then:
         p[q] = p_high[q]
      end if
   end if
end for
for each priority level pl, highest first do:
   if length(queue(pl))>0 then:
      q=queue(pl)
      if p_high[q] < p_low[q] then:
         credit[q] = min(LMs[q],
                         credit[q]+size(head(q)).(1-BWs[q]))
         timerDQ[q] = currentTime+size(head(q))/C
         if credit >= LMs[q] and p[q] = p_high[q] then:
            p[q] = p_low[q]
         end if
      end if
      dequeue(head(q))
      break
   end if
end for
```

Figure 4: PSS algorithm

PSS algorithm also implements the following functions:

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

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 5 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. 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.

```
                            priorities
                           ________________________________
                   queues |                                |\
    Admitted EF--->||-----+ p[AEF]= 1                      | \
                          |                                |  \
  Unadmitted EF--->||-----+ p[UEF]= 2                      |   \
                          |                                |    \
            AF1--->||-----+ p_high[AF1]=3 and p_low[AF1]= 6| PSS ---
                          |                                |    /
            AF2--->||-----+ p_high[AF2]=4 and p_low[AF2]= 7|   /
                          |                                |  /
            CS0--->||-----+ p[CS0]= 5                      | /
                          |________________________________|/
```

Figure 5: 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:

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

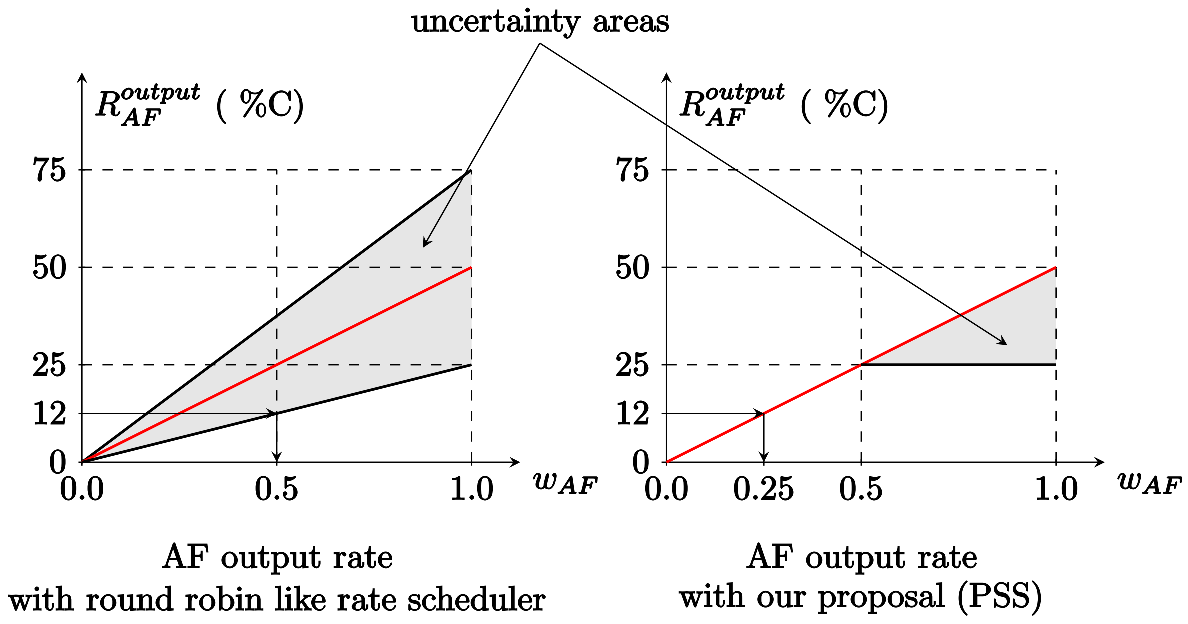

Figure 6: service offered by PSS compared to WRR

As illustrated in Figure 6, 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 Nicolas Kuhn and David Black for reviewing this draft. 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.

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.

[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.

[RFC4594] Babiarz, J., Chan, K. and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006.

[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.

Authors' Addresses

Fred Baker
Santa Barbara, California 93117
USA
EMail: FredBaker.IETF@gmail.com

Anais Finzi
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse, 31400
France
Phone: 0033561338735
EMail: anais.finzi@isae-supaero.fr

Fabrice Frances
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse, 31400
France
EMail: fabrice.frances@isae-supaero.fr

Emmanuel Lochin
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse, 31400
France
EMail: emmanuel.lochin@isae-supaero.fr

Ahlem Mifdaoui
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse, 31400
France
EMail: ahlem.mifdaoui@isae-supaero.fr