An evaluation of software-based TSN traffic shapers using Linux tc

Santiago Torres Borda, Jérôme Ermont

An evaluation of software-based TSN traffic shapers using Linux tc

Santiago Torres Borda, Jérôme Ermont
Université de Toulouse - IRIT - Toulouse INP/ENSEEIHT
Toulouse, France

***Abstract*—In this study, the feasibility of a hardware abstracted switch that targets TSN applications is evaluated. A solution using P4, a flexible solution to program SDN switches, is discussed. Nevertheless, extensions to this approximation have to be made to respect time oriented features that are unavailable to implement in a sole P4 environment. As a consequence, the Linux *tc* package is used to program the traffic control functionalities such as TAS and CBS shapers. The final implementation is a proof of concept that uses *tc* and runs on top of the Linux kernel. Finally, future work is discussed as a possible followup to this project.**

***Index Terms*—TSN, Data plane programming, SDN, Traffic Shaping, tc**

I. Introduction

Networks in general have had a flexibility problem for years[1]. Upgrading or changing any aspect of the network is hampered by the fact that there are a wide variety of switches and routers that make a simple change in the network tedious and impractical. Software Defined Networking (SDN)[13] solves this problem by separating the control plane from the data plane, thus allowing the operation of each switch in the network to be dictated. However, this is not enough to make the network completely flexible. Even though rules can now be specified without having to stop the operation of the network, the data plane is still static and hardware dependent. This is why languages like P4 have been proposed[1]. P4 aims to program the data plane, using the SDN pipeline, so that the user can now specify how they want the frame to be processed at the switch level.

In addition, real-time networks have had a skeptical approach to SDN, as timing constraints are not compatible with the overhead added by these new technologies. Many studies, such as [2], [10], and [11] among others, have attempted to evaluate the feasibility of SDN and P4 as tools for real-time network deployment. Nevertheless, no studies have been conducted in which both tools are deployed and tested for real-time networks.

Moreover, Time Sensitive Networking (TSN) has gained popularity for timed constrained networks by utilizing widely established Ethernet networks[5]. In particular, to reduce latency, they have defined many norms such as 802.1Qav (Credit Based Shaper) and 802.1Qbv (Time Aware Shaper). Where, the first one works on a credit basis by only sending the quantity of bits it has in terms of credits. Whilst, the second one works on a time basis and schedules time constrained and non-time constrained frames into different time slots.

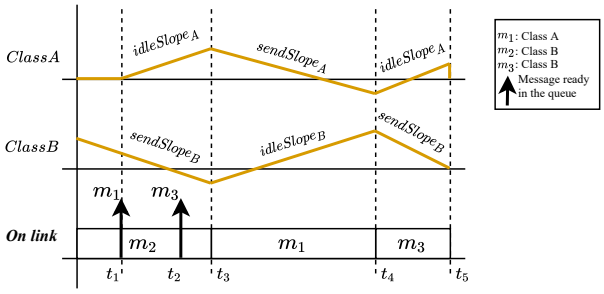

Fig. 1. CBS behavior

The objective of this study will be to propose a clear pathway to a TSN solution within an SDN/P4 environment. In particular, we aim to propose a way to implement bandwidth allocation shapers such as Time-Aware Shaper (TAS) and Credit Based Shaper (CBS) in this environment.

II. Problem statements

A. Traffic shaping in Time Sensitive Networking

TSN norms define tools to adapt Ethernet networks to a Real-Time focused environment[5]. In particular, they allow for synchronization, reliability, latency management and resource management. Where the latency management aspect is mostly composed by traffic shapers that define how frames are scheduled according to their latency requirements. For instance, the conjunction of CBS and TAS is traditionally used in TSN configurations since it allows the consideration of time and event triggered traffic simultaneously.

1) Credit Based Shaper: CBS is a mechanism that aims to limit starvation problems, improve fairness between flows and limit jitter by controlling the output transmission rate. In essence, for a given class a message can only be sent if the quantity of credits, represented in bits, is positive. A class can accumulate credits whenever there's a message in the queue ready to be transmitted. When the queued message is sent, the class' credits are spent until a negative minimum is reached. At that point, the class relinquishes medium access to the next class ready to transmit.

Figure 1 proposes an example of the CBS behavior. Here, three messages are to be scheduled. They are divided between two classes and are shaped by a different CBS. Before t_1, there is no frame in the queue of class A, so the credit stays equal to 0. In contrast, at the beginning of this example,

m_2 is already being transmitted and consequently the credits belonging to class B are being spent. The rate at which the CBS shaped queue is designated to send is called the *sendSlope* and depends directly on the Network's Interface speed.

Once these credits reach a negative minimum, at t_3, no more message from class B can be transmitted. At t_1, m_1 is in the queue of class A. The credit of class A is accumulated until t_3. Where the slope of the credit accumulation is named *idleSlope*. From t_3, m_1 is transmitted and the credits of class A are spent. Finally, the presence of m_3 in class B queue, allowed for it to accumulate credit, which is used to send the aforementioned frame. In this particular case, class' B associated output rate was capped at the value of its associated *sendSlope* when transmitting m_2 and m_3, whereas it was locked to 0 whenever its credit was negative or wasn't transmitting.

2) Time Aware Shaper: Moreover, for TAS, traffic is divided into time slots. For each time slot, only traffic belonging to a certain class is scheduled on the network interface. CDT slots are meant for time constrained traffic, whereas non time constrained traffic is only scheduled during the best effort slot. Finally a guard band is implemented so that best effort slot duration is ensured to not exceed its expected duration. The described functionality is evidenced in figure 2.

B. Data plane programming

Since its conception, Software Defined Networking aims to render networks re-configurable without having to change or reprogram network entities such as switches or routers. First, the control plane was separated from the data plane. As a consequence, a network controller became the sole member of the control plane and dictated routing and switching rules for the whole network[9]. Still, this solution was limited since not every part of the network was programmable. For instance, the data plane was still hardware dependent and would need manual reconfiguration in case the network would accept traffic with new characteristics.

As a solution to this limitation, P4 allowed for the pipeline from figure 3. Compatible with the traditional SDN pipeline, it would allow hardware abstraction by reducing switching and routing features to two functional blocks: a parser and a match+action rule[1]. The first one is executed upon package ingress and fragments the different fields from the packet header. Then, a first match+action rule is performed in which the contents from the header fields are analyzed. Here, an egress rule that indicates where to output the packet can be made. Furthermore, a packet modification stage can be done before having to perform a second match+action rule that

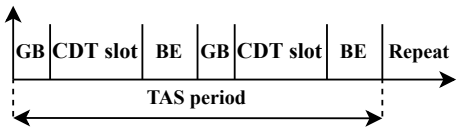

Fig. 2. TAS timeslots

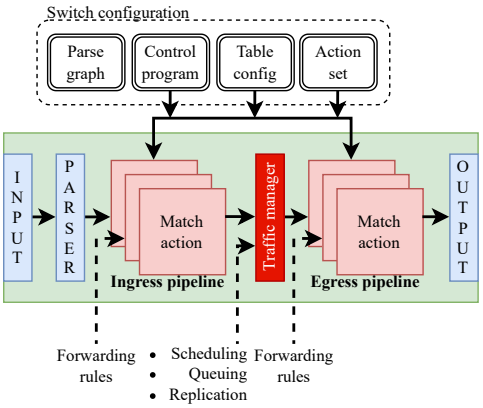

Fig. 3. P4 generic pipeline

would take into account the newly added information. By introducing this pipeline, P4 allows data plane programming in conjunction with control plane programming already established in SDN.

As an example, a TSN compliant P4 switch would go until the traffic management step of the pipeline and then output the packet given no modification is needed after traffic shaping. First, the packet would be parsed and then the analysis VLAN TAG would be carried out. Once the packet's priority is obtained through said tag, the traffic management phase can begin, to then output the packet onto the network.

Every P4 implementation follows a switch model. It can either be specific to a material P4 switch or can be abstract enough to run both on material P4 switches and software switches targets such as DPDK[7] or Open vSwitch[12]. Even though there can be a wide variety of implementations, they all follow the same pipeline from figure 3. The pipeline allows predictable behavior among a wide variety of switches, this way a P4 hardware abstraction layer is defined.

The most general model that can run on a wide variety of targets is called v1model[8]. The model's architecture follows closely figure 3 and is flexible for parsing and match+action rules, yet static for the traffic management features. Since the aim of P4 is to be as abstract as possible, the traffic manager is not programmable by depending on the switch available hardware features, such as the number of queues, interfaces, etc. Given that the traffic manager is in charge of scheduling and queuing, two crucial factors for any TSN shaper implementation, this fact renders a TSN P4 switch implementation infeasible.

III. Motivation

Since an implementation of a TSN P4 switch is deemed impossible, the furthest the switch implementation can go is the ingress parsing, where the switch will only let through VLAN tagged frames. Still, an approximation to achieve TSN functionalities is possible. This implementation can target a software switch so that application layer tools can be used. One such solution would be to target DPDK or Open vSwitch

implementations that run on the Linux kernel which allows the utilization of Linux *qdiscs*[6] to act as a re-configurable traffic manager. By using them in conjunction to the switch mandated rules from P4, a TSN switch mandated by P4 parsing is possible.

IV. Linux Traffic Control

Linux traffic control or tc[6] is a tool from the iproute2 suite used as the network traffic manager for the Linux kernel. The tool's main features include traffic shaping, scheduling, policing and dropping. As a means to process the traffic it employs queuing disciplines or qdiscs, classes and filters. Where qdiscs are the implementation of queuing mechanisms like PFIFO. CBS and TAS work in conjunction to these qdiscs to achieve TSN functionality.

A. Implementing TSN shapers with tc

First, to implement TAS using tc, the *taprio* qdisc[4] must be used. The *taprio* qdisc implements a simplified version of the 802.1Qbv mechanism, which models the different queues by gating mechanisms which open or close at a specific point in time. Thus, allowing to schedule a specific set of queues in a given time slot. As an effect of this implementation, the periodic guard band is not present.

Second, to implement CBS using tc, the *cbs* qdisc must be used alongside the *mqprio* qdisc. Where the latter is used to map outgoing traffic to a specific priority. The *cbs* qdisc is then set to a specific queue by defining its *idleslope*, *sendslope*, *locredit* which is the negative minimum amount of credits allowed and *highcredit* which is the maximum amount of credits allowed. These values are inherent to the network interface speed and can be calculated as defined in [3].

B. Traffic shaping with tc

In order to prove the feasibility of a P4 switch that offloads the traffic management to *tc's qdiscs*, two tests were defined on the network configuration shown in figure 4. The first one implements two queues shaped by a Time Aware Shaper. On one end of the switch, a host is emitting frames as fast as it can, alternating the priority at each emission. On the other side, another host listens for those frames and registers the timestamps at which those frames arrive. This way traffic belonging to the two priorities was generated regardless of the ongoing time slot. Similarly, the Credit Based Shaper is set to shape one of two queues, where the second queue is free to transmit as soon as it can. Here, the emitter sends with the same priority many concurrent frames so that the throughput is affected by the presence or absence of credits. On the receiver side the perceived throughput for both priorities is registered. For every test made, the traffic was set to UDP in order to emulate real-world behavior.

Since our goal is to test the feasibility of *tc* as a TSN-oriented traffic manager, the switch is reduced to a bridge between two interfaces. In addition, since *tc* runs on the application layer, the traffic priority found in the frames is mapped to a destination port instead, where 7777 represents

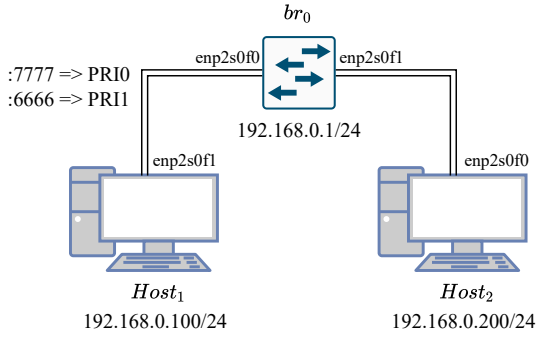

Fig. 4. Proposed test bed

in both tests the time sensitive data and 6666 the best effort data.

Because *tc* runs on the application layer, tools for fast and easy prototyping that work from the application layer are used. For instance, packet generation was done using Netcat and iPerf3. The first one allowed for packet creation having a specific length and destination port whereas the second allowed to generate simultaneous concurrent traffic using the same destination port. This allows to set the testing environments for tc driven TAS and CBS respectively.

V. Results

A. Time Aware Shaper

The TAS evaluation used the test bed from figure 4 to obtain the behavior shown in figure 2. Here an arbitrary CDT slot of $80ms$ and a BE slot of $20ms$ for a total TAS period of $100ms$ was set. Although arbitrary, these testing conditions allow TAS shaping characteristics in figure 5 to be observed.

Figure 5 is divided in two parts. The bottom plot shows the frame emission and the top plot shows frame reception. As stated in section IV-B, the frames are emitted as fast as

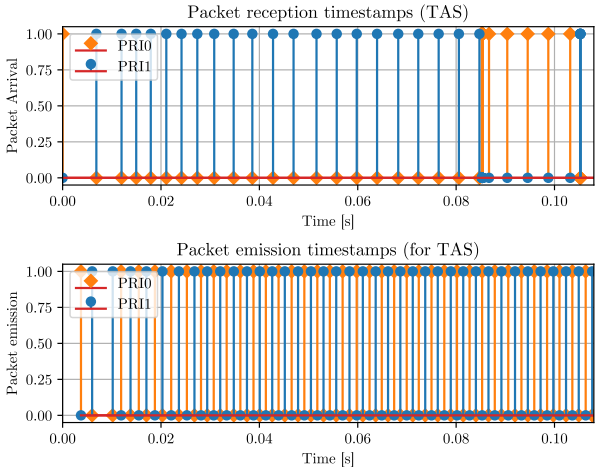

Fig. 5. Time Aware Shaper experiment results

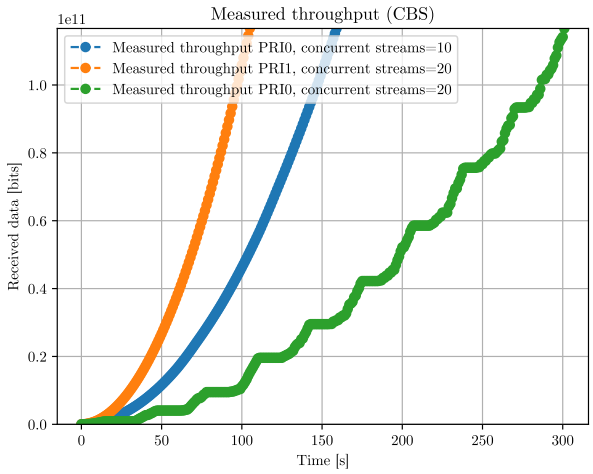

Fig. 6. Credit Based Shaper Experiment results

possible by $Host_1$. It is evident from the plot on the top that the scheduler, as expected, divides the traffic into their respective slots. Nonetheless, slot duration is not respected by adding consistently $2.05ms$ accounting by the *tc* added overhead and the frame transmission delay. This behavior is expected since the Linux kernel used has no Real-Time features.

B. Credit Based Shaper

Similar to the Time Aware Shaper experiment, the Credit Based Shaper experiment result from figure 6 yields the expected output. Whenever the $PRI0$ queue is congested, its throughput will be 0 for as long as the *idleslope* accumulates the credit back.

In this test, we set the CBS to be using half of the available bandwidth at all times by configuring the *idleslope* to 50Mbps which is half of the network's interface output speed. Consequently, the *sendslope* was set to $-50Mbps$, the *hicredit* to $6Mbits$ and the *locredit* to $-500kbits$. Here, concurrent streams were generated periodically with the separation of a second per set of concurrent streams. This allowed the visibility of CBS's behavior observed in figure 6.

VI. Conclusions and Perspective

In this paper it was concluded that P4 is not abstract enough to program a TSN oriented switch. Nevertheless, it can work in conjunction with other software implementations such as *tc* and DPDK or Open vSwitch in order to reach the needed modularity. It was also shown that *tc* is a well implemented alternative to P4's static traffic manager and is able to implement scheduling TSN features with a considerable delay that can be lowered with a real time implementation of the Linux kernel.

As future work, implementing P4 in conjunction with *tc* and evaluating it as an SDN deployable configuration could be of interest. Furthermore, if said solution isn't enough in terms of expected time constraints, a new Hardware Abstraction Layer could be proposed for switch programming that would allow to integrate hard Real-Time features into the data plane.

References

[1] P. Bosshart et al, "P4: Programming Protocol-Independent Packet Processors," SIGCOMM Comput.Commun.Rev., vol. 44, (3), pp. 87-95, 7, 2014.
[2] Y.-W. Chen, L.-H. Yen, W.-C. Wang, C.-A. Chuang, Y.-S. Liu, and C.-C. Tseng, "P4-enabled bandwidth management," in 20th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2019, pp. 1–5.
[3] V. Costa, *CBS - credit based shaper (CBS) qdisc*, Linux Man Page, 2017.
[4] V. Costa, *TAPRIO - time aware priority shaper*, Linux Man Page, 2018.
[5] Time-sensitive networking (TSN) task group, "TSN Standards." [Online]. Available: https://1.ieee802.org/tsn.
[6] B. Hubert, *tc - show / manipulate traffic control settings*, Linux Man Page, 2001.
[7] Intel Corporation. "Intel DPDK: Data plane development kit." [Online]. Available: https://www.dpdk.org/
[8] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," IEEE Access, vol. 9, pp. 87094–87155, 2021
[9] N. McKeown et al, "OpenFlow: Enabling Innovation in Campus Networks," SIGCOMM Comput.Commun.Rev., vol. 38, (2), pp. 69-74, 3, 2008.
[10] R. Rotermund, Timo Häckel, P. Meyer, F. Korf, and T. C. Schmidt, "Requirements analysis and performance evaluation of SDN controllers for automotive use cases," in 2020 IEEE Vehicular Networking Conference (VNC), 2020, pp. 1–8.
[11] D. Thiele and R. Ernst, "Formal analysis based evaluation of software defined networking for time-sensitive Ethernet," in 2016 Design, Automation Test in Europe Conference Exhibition, 2016, pp. 31–36.
[12] The Linux Foundation. "Production quality, multilayer open virtual switch." [Online] http://www.openvswitch.org/.
[13] D. Kreutz et al, "Software-Defined Networking: A Comprehensive Survey," Proc IEEE, vol. 103, (1), pp. 14-76, 2015.