Towards Approximate Fair Bandwidth Sharing via Dynamic Priority Queuing

Carmelo Cascone[*‡], Nicola Bonelli [†], Luca Bianchi[*], Antonio Capone[*], Brunilde Sansò [‡]
[*] Politecnico di Milano, Italy
[†] Università di Pisa, Italy
[‡] Ecole Polytechnique de Montréal, Canada

***Abstract*—We tackle the problem of a network switch enforcing fair bandwidth sharing of the same link among many TCP-like senders. Most of the mechanisms to solve this problem are based on complex scheduling algorithms, whose feasibility becomes very expensive with the line rate of today switch ports (10-100Gb/s). We propose a new scheme called FDPA in which we do not modify the scheduler, but instead we use an array of rate estimators to dynamically assign traffic flows to an existing strict priority scheduler serving only few queues. FDPA is inspired by recent advances in programmable stateful data planes, as such we propose a design that is amenable with current abstractions such as P4 or OpenFlow. We conducted experiments on a physical 10Gb/s testbed. We present preliminary result showing that FDPA produces fairness comparable to approaches based on scheduling.**

I. Introduction

It has been reported that TCP traffic represents 80-90% of the packets and bytes flowing today in the Internet [6]. It follows that most of the traffic sources adapt their sending rate according to the perceived available bandwidth. Indeed, TCP is the instantiation of an important design choice that contributed to the success of the Internet: to leave congestion control to the end-systems, thus permitting a relatively simpler implementation of the interconnection devices. TCP rate control algorithms, such as Additive-Increase-Multiplicative-Decrease (AIMD), help in maintaining a fair allocation of network resources on a *per-flow* basis. In the simplest case of multiple TCP streams, all experiencing the same RTT and sharing the same FIFO queue, each flow tends to occupy the same portion of the link bandwidth [19].

However, relying only on end-systems to guarantee fairness is not enough due to ill-behaving users and issues intrinsic to TCP-like algorithms. Example of such situations of unfairness are: (i) applications that open a large number of parallel TCP connections, e.g. peer-to-peer, or that tweak TCP to get better performances; (ii) non-TCP-like protocols, i.e. protocols that do not respond to congestion signals such as drops, and (iii) the dependence of standard TCP to the round-trip times (RTT) [19].

For these reasons, most Internet service providers (ISPs) tend to enforce direct control over their customers, by throttling their traffic at the network edge, limiting the maximum bandwidth of each user to a feasible, but *static* network allocation. This approach allows ISP to leave their core and interconnections with other ISPs uncongested at all the time. The downside of such static bandwidth allocation is that the excess bandwidth remains unused, even in common situations of very low usage, such at night.

Researchers have proposed solutions that enforce a more *dynamic* bandwidth allocation in the network interconnection devices. In these approaches, instead of capping the maximum sending rate at all times, network devices are able to redistribute the unused capacity (if any) to those users asking for more. The trick here is to design a bandwidth enforcement scheme that (i) guarantees that all users can obtain at least the level of service they paid for, i.e. minimum rate guarantees, and (ii) when unused capacity is available, that is shared by all users, with no one prevailing on others. This should be performed on a link of 10-100 Gb/s.

Fair Queuing (FQ) scheduling [9], [16] is the textbook approach to enforce almost perfect fairness among different traffic sources, independently of the behavior of the end-hosts. A switch implementing FQ works by assigning users to different queues, where a user is an arbitrary aggregate of packets, e.g. with the same IP source address or the same TCP/UDP 5-tuple. FQ provides high precision of bandwidth partitioning, but unfortunately, such precision comes at a considerable expense: (i) the time to process a packet depends on the number N of active users, precisely $O(\log(N))$; and (ii) N per-user queues are required.

The first limitation is important with today's throughput requirements which drastically reduce the maximum processing time allowed for a packet, e.g. a switching chip with aggregate throughput of 1 Tb/s has a time budget of 1 ns to process a minimum size packet. The second limitation affects switching hardware implementations. Here the number of queues impacts both the memory requirements and the combinatorial logic necessary to implements the circuitry of the scheduler[1]. As a consequence, it is hard to scale FQ implementations to hundreds, thousands or more users. For this reason the number of queues available in commercial hardware switches is usually bounded to less than 10 [1]. This consideration is also at the base of legacy quality of service (QoS) approaches such as DiffServ, where traffic is aggregated into few classes.

[1]For a scheduler to be work-conserving, i.e. to serve a packet if at least one can be served, all N queues must be examined at the same time. As such, the number of wires to implement such a structure depends on N.

In this work, our focus is to devise a design for a bandwidth enforcing a scheme in which both time and implementation complexity do not depend on the number of active users N.

This work is inspired by recent advances in Software-Defined Networking (SDN) and data plane programmability. Emerging abstractions such as P4 [5], OpenState [2], OPP [3], FAST [15], and Domino [22] allow network operators to perform flexible stateful packet processing inside the network. The statefulness of the aforementioned approaches, lays in the ability to program forwarding rules that read and modify data plane's forwarding state. Based on this capability, a number of studies have been published, showing how to implement existing and new forwarding functions using programmable data planes [8], [12], [20], [24].

We follow this path and design a scheme to enforce fair bandwidth sharing that is amenable with programmable data plane abstractions. To this purpose we do not modify the scheduler, instead we use a widely-deployed strict priority (SP) scheduler with only few queues. We enforce fairness by dynamically assigning priorities to users according to the sending rate history of those. We call our design FDPA (Fair Dynamic Priority Assignment). In FDPA, packets belonging to a user whose arrival bitrate is equal or less than its fair share are given priority over those users generating traffic at higher rates. FDPA does not provide precise bit-level or packet-level fairness, but it approximates a fair repartitioning over longer timescales, in the order of few RTTs.

The scalability of FDPA does not depend on the number of queues, but instead on the number of rate estimators that can be instantiated in the switch. Precisely, while the circuitry to implement a rate estimator can be shared among many flows[2], the switch is required to maintain per-user state, i.e. the measured rate. Hence, the only limit of FDPA is the memory available in a switching chip.

In this work we address the feasibility of the FDPA approach by performing experiments on a 10 Gb/s testbed using a software prototype implementation. Results show that FDPA produce fairness comparable to other schemes such as Deficit Round Robin (DRR). We found that such an approach produce a trade-off between fairness and throughput, in which one or the other are penalized.

To summarize, the contributions of this paper are:

- Design of FDPA, a scheme to enforce approximate fair bandwidth sharing. Switch requirements to support FDPA are a (i) strict priority scheduler and (ii) the ability to manage data plane's state to measure the arrival bitrate of each user.
- Evaluation of FDPA and other Linux's rate-control schemes using a 10 Gb/s testbed with real TCP traffic.

We begin by reviewing the related work in Section II, we then introduce the FDPA design in Section III and discuss its implementation options with programmable data planes. In Section IV we present the experimental results from the 10 Gb/s testbed, before concluding with a discussion on open questions and future work in Section V.

[2]In a typical pipelined hardware architecture, that would be a stage of the pipeline.

II. Related work

To reduce implementation and time complexity of FQ, a number of algorithms have been proposed in the literature. Deficit Round Robin (DRR) [21] is probably the most known and widely-deployed algorithm. DRR that was proposed to address the time complexity of FQ achieving $O(1)$ execution time per packet. However, DRR still requires per-user queues.

To overcome DRR's limitations, further approximations have been proposed. Stochastic Fair Queuing (SFQ) [13] is a probabilistic variant of FQ. Here traffic streams are hashed onto a smaller number of queues, and the hash function is periodically perturbed to minimize the time where 2 users collide onto the same queue. Here the quality of the approximations depends on the number of queues, and the perturbation interval. Finally, Approximate Fair Dropping (AFD) [17] employs a form of active queue management (AQM) by dropping packets before being stored on a simple FIFO queue. Dropping decisions are based on the recent history of packet arrivals, with higher probability of drop for users sending at higher rates. AFD has been used in several switch and router platforms at Cisco Systems [18].

Our approach shares the same design principles of AFD: (i) avoid using per-user queues in favor of per-user soft state, and (ii) achieve bandwidth partitioning by opportunistically dropping or delaying packets rather than by enforcing rate by using scheduling. However, while the AFD design allows for an efficient implementation in a fixed-function ASIC, its realization with programmable data plane primitives might not be straightforward. Specifically, AFD requires the implementation of a shadow buffer in which packets are removed at random. We are not aware of any data plane abstraction providing native support for such data structure. Its behavior could be approximated using other primitives, however this would require a dedicated study. Instead, we prefer to explore the feasibility of FDPA, as we will discuss in Section III-B, it requires much simpler primitives already exposed by current data plane abstractions.

Finally, a more recent approach named PIFO has been proposed to address the need of a programmable scheduler [23]. However, similarly to fixed-function schedulers, in PIFO the number of distinct flows that can be served with a fair queuing discipline is bounded by the number of queues. In their proposed design, such bound is 2048 in total or 32 per port in a 64 port switch. While one could imagine of dedicating all 2048 queues to a port, the authors do not provide any evaluation of their scheduler with realistic traffic traces.

III. FDPA Design

In this section we describe the design of a packet forwarding pipeline implementing FDPA. For the sake of explanation and without loss of generality, we assume a switch with rate controlled only on one egress port.

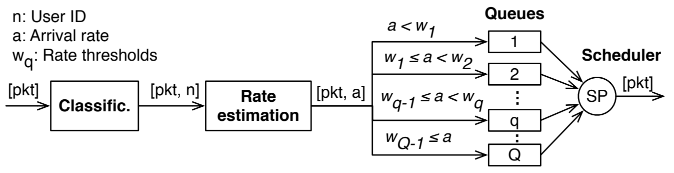

Fig. 1. FDPA forwarding pipeline.

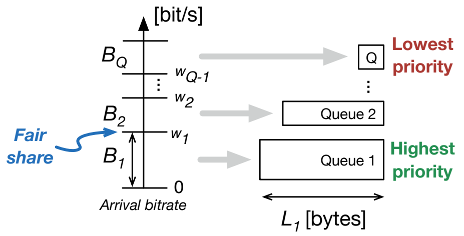

Fig. 2. Rate bands and queue size in FDPA

Figure 1 depicts the design of the pipeline. Packets are first classified per user and then processed by a rate estimator which measures the arrival bitrate of the specific user. Packets are then stored in one of the Q priority queues such that the higher is the arrival rate, the lower will be the priority. A strict priority scheduler (SP) serves queues in priority order: packets of priority q are dequeued only if all other queues with higher priority are empty, where $q = 1$ is the highest priority.

The measured arrival rate for a given user at a given point in time, determines an active "band" for that user. Packets arrived in band B_q will be assigned with priority q (Figure 2). The first band B_1 represents the minimum guaranteed portion of the link capacity allocated to each user, for this reason we require that $N \times B_1 <= LinkCapacity$. Moreover, to further penalize ill-behaving users, each queue has a different size L_q, with smaller values for low priority queues.

A. Rationale

To discuss the rationale behind this design, we begin with the case of a scheduler with only 2 queues ($Q = 2$), high priority and low priority; we then explain the need of more queues.

Two priorities. When congestion occurs, users sending below their fair share are prioritized against others sending at higher rates. Given the limited size of the queues, packets with low priority are delayed and in the worst case of a full buffer, dropped upon arrival. Such an event signals the TCP source to reduce the transmit rate. With FDPA, this reduction is expected to continue until the transmit rate hits the first band, in which case the user is prioritized again. Assuming that all sources are TCP-like and produce long-lived flows, under severe congestion we expect traffic sources to shape their transmit rate around their fair share, i.e. the upper threshold of the first band. Swapping queues frequently, can cause packet reordering at the receiver, confusing TCP congestion control and affecting throughput. We are interested in measuring this effect when using FDPA.

In the case of non-elastic sources, e.g. constant bitrate, B_1 represents the maximum rate that a source can send with

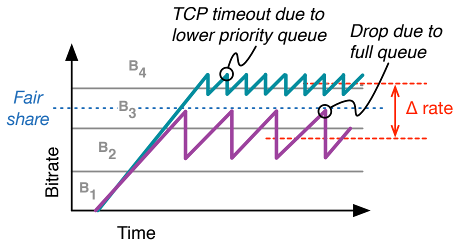

Fig. 3. Example of 2 TCP sources competing for the excess bandwidth when using more than 2 priorities.

guarantees of bounded latency and minimum drop probability. Indeed when a user hits the first band, packets are always served by the same, maximum priority queue, hence preventing disruption from other TCP sources when aiming to transmit at higher rates.

However, if some sources are using less than their fair share or because not all the link capacity has been reserved, i.e. $N \times B_1 < LinkCapacity$, using only 2 priorities does not enforce equal distribution of the excess bandwidth. Indeed, if we assume that capacity has been allocated for many users, but only few of them are active and sending TCP traffic, we can expect that those users will be competing in the same low priority FIFO queue, without any guarantee of fairness.

More priorities. To enforce equal distribution of the excess bandwidth, we need to introduce more priorities, such that the more a source increases its rate, the less priority it will get with respect to other users. When all sources are TCP-like, following the same rationale of the previous case, we expect the transmit rate of each user to converge to a fair share that considers the excess bandwidth. Such fair share will lay in a rate band other than B_1

Figure 3 illustrates the expected behavior of 2 TCP-like sources competing for the excess bandwidth. In this example, one source (1) is ill-behaving as it uses a more aggressive rate control algorithm[3]; the other source (2) is well behaving, as for each congestion signal it halves its transmit rate. At steady state, both sources tend to share the same queue with priority 3, however the different rate-control behavior that they implement causes them to oscillate around different average values. Indeed, (1) always tends to increase its rate until it falls in the 4th band, which cause its packets to timeout as the scheduler will spend as much time as needed to serve packets of higher priority; (2) instead has higher drop probability when it falls in band B_3, as here the queue is monopolized by packets of (1). However, by always assuring an higher priority for lower rates, increase of (2) are always guaranteed at least until the lower threshold of band B_3. Intuitively, we expect that the difference between the average transmit rate of both sources ($\Delta rate$) will be smaller with narrower bands, hence producing a more fair allocation.

Unfortunately, as in the case with only 2 priorities, we except that multiple narrower bands will increase the risk of

[3]The behavior of this source is similar to the case of a user opening multiple TCP streams.

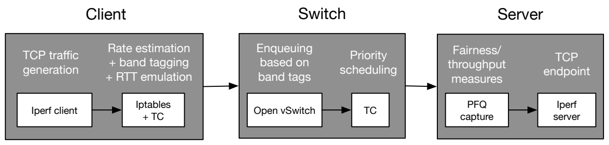

Fig. 4. Software-based processing pipeline used in experiments.

packet reordering, affecting the overall throughput. We are interested in measuring such a trade-off between fairness and throughput.

B. Implementation with programmable data planes

Classifying packets per user is easy and can be done using a match-action table as defined by OpenFlow [14] or P4 [5]. Using such tables one can match on specific header fields and write the corresponding user ID n on the packet's metadata.

Estimating the bitrate of a flow might be tricky at high speed. In the simplest case, the switch needs to maintain for each user a byte counter and a timestamp of the last time the rate estimation was updated. Updates of the rate values are triggered by packets arrival if the timestamp of the packet exceeds a predefined interval, i.e. the minimum interval over which the average bitrate is evaluated. The rate is then computed dividing the number of bytes by the interval between the packet's timestamp and the stored timestamp. While division is an operation that might be hard to perform in a line rate switch, in [20] it is shown how this operation can be approximated with good precision using primitives available in programmable data planes. A second match-action table can then be used to direct packets to the different queues according to the estimated rate band, written in the packet's metadata.

Along with programmable data planes, we notice how FDPA can be implemented in switches supporting OpenFlow v1.3+. Indeed, OpenFlow define "meters" that can be configured with different bands as defined by FDPA, such that packets hitting a given rate can be marked using the the DSCP field.

Finally, priority schedulers are a common structure available today in switching hardware.

IV. Experimental results

We now evaluate the feasibility and performance of FDPA using a software-based prototype implementation. We are interested in measuring the effects of different bands assignment on both fairness and throughput. We also compare FDPA with other approaches such as DRR.

A. Testbed

We used 3 desktop machines with 8-core Intel Xeon E51660V3 CPUs (3.0GHz), equipped with multiple Intel 82599 10G NICs. One machine acts as a switch with 4 10G ports, another machine is used to generate traffic from 2 ports, while the last is used to both generate and receive traffic form different ports. Each machine runs a Debian 9.0 Stretch based on a Linux Kernel v4.9.16.

Figure 4 shows the processing pipeline used to emulate FDPA. We use `iperf` to generate TCP traffic, Linux's `iptables` to estimate the rate and tag packets accordingly. In our design, rate estimation should happen in the switch, however, to simplify the prototype implementation we decided to move it to the client machines. We use Linux's `tc` (Traffic Control) to emulate different RTTs at the clients and to perform priority scheduling at the switch. Open vSwitch is used to steer packets to the different queues based on the band tags. Finally, we use PFQ [4], a framework for accelerated packet I/O, to measure the bitrate of each user. Both clients and server use TCP Cubic, with the default parameters found in the Linux Kernel v4.9.16. We only adjust the memory available to TCP buffers to allow for a large number of connections. We set the MTU of all interfaces to 1500 bytes.

We configure sources to experience an emulated RTT of around 5 ms with maximum 0.25 ms of variable jitter with 25% correlation. TCP increases its sending rate at RTT timescales, hence for FDPA to promptly respond to rate variations, the estimation interval should be in the order of few RTTs. For this reason we use estimations intervals of around 30 ms.

B. Metrics

We measure the quality of an experiment using two metrics: (i) the aggregate throughput (TPut) normalized over the link capacity, i.e. bounded between 0 and 1, and (ii) the Jain's Fairness Index (JFI) [10]. The JFI is a popular fairness measure defined as:

$$JFI = \frac{(\sum_n x_n)^2}{N \cdot \sum_n x_n^2}$$

where x_n is the normalized rate of a user n and N is the total number of users. The normalized rate x_n is defined as:

$$x_n = \frac{MeasuredRate_n}{FairRate_n}$$

In our experiments each user is assigned with the same fair share. The JFI is bounded between 0 and 1, where 1 is a fair distribution and 0 is a discriminating one.In testing FDPA we aim at maximizing both TPut and JFI.

C. Results

Figure 5 shows the results obtained from the experiments. We generate long-lived TCP from a varying number of users, 50, 100 and 200[4], and varying the number of TCP connections per user based on 4 scenarios: (i) all users have the same number 1 of TCP connections, i.e. they all well-behave, (ii) when 1/4th of the users mis-behave by opening 10 parallel TCP connections, (iii) when half of the users mis-behave, and (iv) when each user has a number of connections uniformly distributed between 1 and 10.

When testing FDPA, we vary the number and size of rate bands. We use the following notation to describe an FDPA configuration: $F(FirstBand + NumBands * BandSize)$, where $FirstBand$ is the size of B_1, $NumBands$ is the number of bands following the first one, each one of

[4]We put a limit to 200 as we noticed that our experimental setup suffers of performance degradation when emulating more users

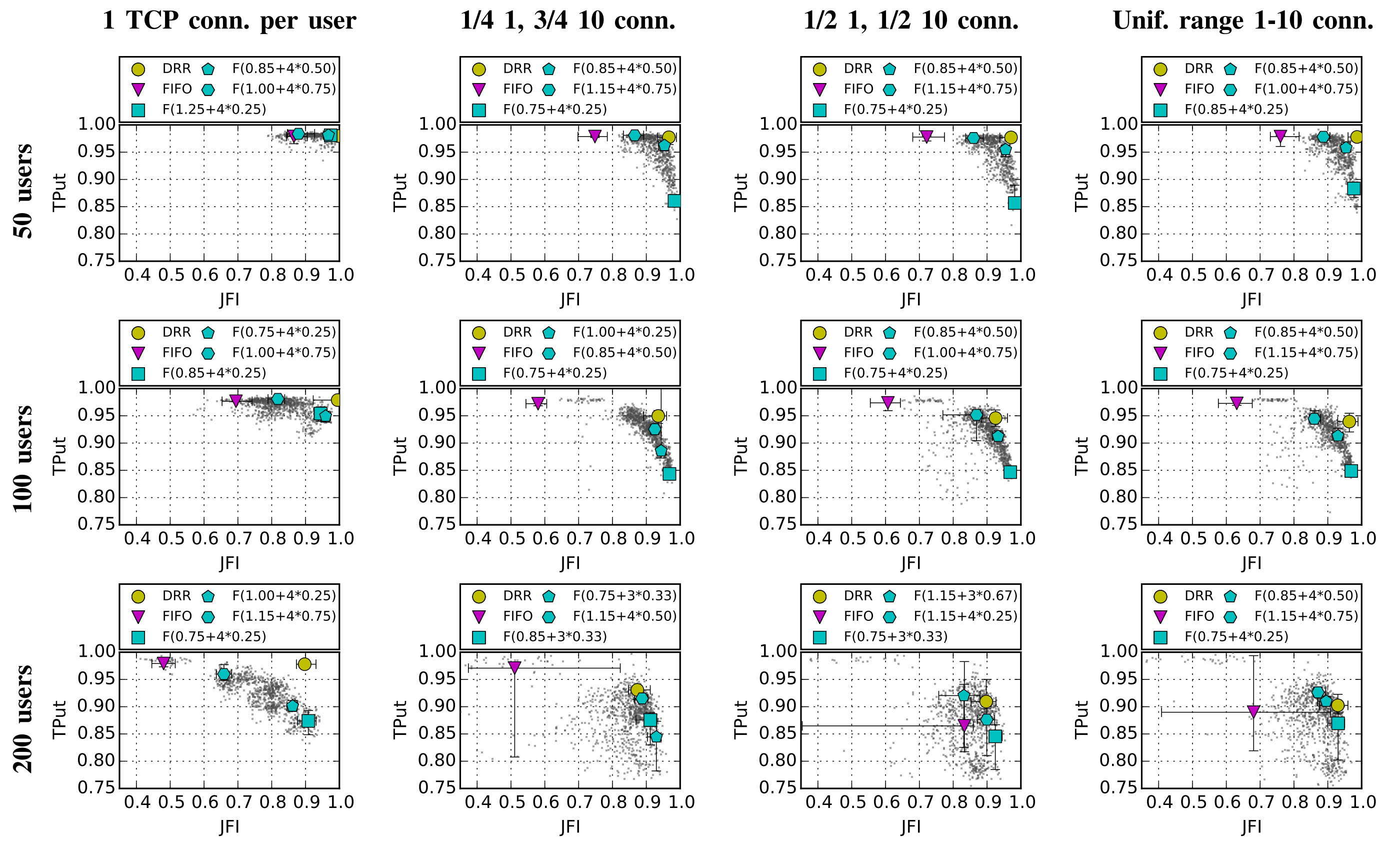

Fig. 5. Experimental results

size $BandSize$, except for the last one that has infinite size, i.e. up to the link capacity. $FirstBand$ and $BandSize$ are expressed as a proportion of the fair share, e.g. $F(1 + 4 * 0.5)$ describes a configuration where the first band is exactly the fair share, and the other 4 bands have size half of the latter. We perform experiments with $FirstBand \in \{0.75, 0.85, 1, 1.15, 1.25\}$, $NumBands \in \{3, 4\}$ and $BandSize \in \{0.25, 0.33, 0.50, 0.67, 0.75\}$. For the queue size we found that the following sizing provides optimal performances: $L_q = \min(20, BDP/q^q)$, where BDP is the bandwidth delay product $\overline{RTT} \times LinkCapacity$. With $\overline{RTT} = 5ms$, the sizing for 5 queues is $L_1 = 4166$ (MTU-size packets), $L_2 = 1041$, $L_3 = 154$, $L_4 = 20$, and $L_5 = 20$.

At the server, we collect samples of the average bitrate over a 1-second interval, each second at the same time for all sources, for 50 seconds. We start sampling 30 seconds after starting `iperf`, allowing all TCP sessions to converge to their average bitrate. For each second, we then compute both the JFI and TPut. In the plots, we show the median of the JFI and TPut samples for each experiment, along with a 80% confidence interval. For each traffic scenario, we plot only three configurations of FDPA, the one with the best TPut, the one with the best JFI, and the one that maximize the product of both. We also provide a scatter plot of all JFI and TPut values obtained in all FDPA configurations. This explicitly shows a clear trade-off between TPut and JFI.

Finally, we compare results with the following cases:

FIFO. All users are served using 1 FIFO queue of size $L = BDP$, e.g. 4166 MTU-size packets with $\overline{RTT} = 5ms$. This is our worst case, when fairness is not enforced.

DRR. The switch implements DRR scheduling with per-user queues. We use the `tc-drr` implementation provided as part of the Linux's `tc` suite. We use DRR as the best case scenario, however it is important to note that while it is still feasible to provide a large number of per-user queues in software, the same does not apply to hardware switches, where an a priori instantiation of hardware resources (memory and logic circuitry) is required. Remember that the majority of today switching chips provide 10 or less output queues per port [1].

As expected, FDPA holds the promise of enforcing fairness w.r.t. a single FIFO queue in all scenarios, producing results comparable to the ideal case of a DRR scheduler with per-user queues. However, fairness comes at the cost of throughput. We observe how configurations of FDPA that use narrower bands provide more fairness, between 0.95 and 0.99 in most cases. Unfortunately, these configuration systematically incur in throughput degradation, down to 0.85 in some cases, when for the same scenario DRR achieves almost perfect fairness with throughput comparable to that of a FIFO queue, i.e. optimal around 0.98, or little less around 0.95. Vice versa, larger bands improve throughput, at the expense of fairness.

V. Discussion

How to improve throughput? Preliminary analysis show that throughput degradation is mostly caused by packet reordering due to frequent changes in the queue assignment, which confuses the TCP congestion control. The problematic part, is when users are prioritized again. Here a burst of consecutive packets swapped from a low priority queue to an higher priority one, with the effect of having subsequent packets being transmitted before those arrived earlier. A solution to this problem could be that of using a "flowlet-based" approach [11], in which queue assignments are valid for the whole burst of packets, where bursts are separated by an idle time usually comparable to the RTT. This would increase the probability of having all packets from the low priority queue sent before the arrival of the new burst. Detecting flowlets is a common function implemented by stateful data plane abstractions [7], [12], [20]. We leave exploring such a more advanced design for future work.

Rate estimation. An alternative to the average estimators that we used are token buckets-based estimators. The advantage of token buckets lays in their ability to immediately respond to rate spikes and bursts of packets, while an average estimator leaves enough time to aggressive user to congest the first queue. We know that the downside of frequent band variations correspond to higher risk of packet reordering, and preliminary results on our testbed using token buckets show that this is the case. However, we believe that using token buckets along with per-flowlet queue assignment could help in improving both fairness and throughput. We leave this for future work.

How to compute the fair share? We envision an external controller (or switch-internal control plane) that periodically adjusts band sizes by counting the number of active users. In the case of a service provider network, where the number of active users varies slowly, we do not expect that the frequency of the estimation process might be a limit for the scalability of the approach. Indeed, using many priorities helps in absorbing temporary rate spikes and minor variation of the fair share. How to efficiently implement user estimation is outside the scope of this work, however, we note that a controller could use the same counter instantiated at the switch for the rate estimation process.

Latency. Similar works on fair scheduling and active queue management aims also at reducing the average time in queue for the packets. In our approach, by dedicating only 3-4 queues to FDPA, if one wants to improve latency for a certain traffic portion, that should be sent to a dedicated highest priority queue, while the first band of FDPA should be assigned to a lower one. The same approach is used in Cisco's implementations of AFD [18].

VI. Conclusions

We introduced FDPA, a design for a packet forwarding pipeline to enforce approximate fair bandwidth sharing among many users sharing the same link of fixed capacity. FDPA is based on existing abstractions for stateful programmable data planes. We performed experimental evaluation on a 10 Gb/s testbed. Results show that performance are close to that of an ideal DRR scheduler with dedicated queues per user (50, 100, 200). We identified a trade-off between fairness and throughput, in which throughput is penalized when configuring FDPA for more fairness. Preliminary analysis show that packet reordering is the cause of such effect. We identified potential solutions to such problem that we leave for future works.

References

[1] Packet buffers. https://people.ucsc.edu/~warner/buffer.html.

[2] G. Bianchi, M. Bonola, A. Capone, and C. Cascone. Openstate: Programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM CCR 44.2*, 2014.

[3] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone. Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing. *CoRR*, abs/1605.01977, 2016.

[4] N. Bonelli, S. Giordano, and G. Procissi. Network traffic processing with pfq. *IEEE JSAC*, 34(6), 2016.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR 44.3*, 2014.

[6] CAIDA. Analyzing UDP usage in Internet traffic. https://www.caida.org/research/traffic-analysis/tcpudpratio/, 2009.

[7] C. Cascone, L. Pollini, D. Sanvito, and A. Capone. Traffic management applications for stateful sdn data plane. In *EWSDN*, 2015.

[8] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sansò. Fast failure detection and recovery in sdn with stateful data plane. *International Journal of Network Management*, 27(2), 2017.

[9] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM CCR 19.4*, 1989.

[10] R. Jain, D.-M. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. *CoRR*, 1984.

[11] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM CCR 37.2*, 2007.

[12] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. HULA: Scalable load balancing using programmable data planes. In *ACM SOSR*, 2016.

[13] P. E. McKenney. Stochastic fairness queueing. In *IEEE INFOCOM*, 1990.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM CCR 38.2*, 2008.

[15] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan. Flow-level state transition as a new switch primitive for SDN. In *ACM HotSDN*, 2014.

[16] J. Nagle. On Packet Switches With Infinite Storage. IETF RFC 970, 1985.

[17] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM CCR 33.2*, 2003.

[18] R. Pan, B. Prabhakar, F. Bonomi, and B. Olsen. Approximate fair bandwidth allocation: A method for simple and flexible traffic management. In *IEEE Communication, Control, and Computing, 46th Annual Allerton Conference on*, 2008.

[19] L. Qiu, Y. Zhang, and S. Keshav. Understanding the performance of many TCP flows. *Computer Networks*, 37(3–4), 2001.

[20] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *USENIX NSDI*, 2017.

[21] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking*, 4(3), 1996.

[22] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *ACM SIGCOMM*, 2016.

[23] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *ACM SIGCOMM*, 2016.

[24] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *ACM SOSR*, 2017.