Fair dropping for multi-resource fairness in software routers: extended abstract

Vamsi Addanki, Leonardo Linguaglossa, James Roberts, Dario Rossi
Telecom ParisTech, France

This is the extended abstract for a proposed demo. We demonstrate that fair dropping is an effective means to realize fair sharing of bandwidth and CPU in a software router. Analysis underpinning the effectiveness of the proposed approach is presented in [1].

1 MULTI-RESOURCE FAIRNESS IN SOFTWARE ROUTERS

Fair sharing of *bandwidth* between 5-tuple flows is a classical issue in networking that has received considerable attention since Nagle's pioneering proposal in RFC 970. To impose fair shares in the network notably brings robust and predictable network performance and relieves end-systems from implementing a standard, approved congestion control, when applications might have different requirements.

As software routers are increasingly called to implement a range of virtualized network functions and their processing capacity is adjusted dynamically to meet current demand, *additional* bottlenecks can arise that limit flow rates. We concentrate here on the CPU bottleneck. CPU sharing is complicated by the fact that flows can have widely differing per-packet processing requirements depending on the functions they require.

We follow Ghodsi *et al.* [3] in supposing routers should simultaneously provide fair shares of bandwidth in bit/s and CPU in cycles/sec. However, rather than realizing dominant resource fairness as in [3], we impose max-min fairness independently on each resource to yield the form of bottleneck-based fairness advocated by Bonald and Roberts [2].

2 NEED FOR A NEW APPROACH

To attain required high forwarding speeds, software routers use a number of optimizations that make it difficult to implement classical fair queuing schedulers [6]. Significant examples are batch IO, to reduce interrupt pressure in kernel bypass stacks, and batch processing, to maximize hits in the instruction cache and to minimize the overhead of processing graph traversal. On the other hand, high-speed software routers are intrinsically flow-aware, since load balancing is performed using a hash of header fields (typically the usual 5-tuple), computed in the NIC and appended to packet metadata; this facilitates per-flow processing.

Rather than trying to adapt algorithms like start-time fair queuing or deficit round robin, we maintain the current packet forwarding scheme (namely, FIFO ring buffers with polling and batch service) and add a fair dropping function to reduce flow rates as necessary to achieve max-min fairness. Of course, packet dropping has previously been used as an alternative to scheduling to share *bandwidth*. Our approach takes account of the specifics of the software router and is more precise that previous proposals but is not fundamentally different. The main originality of our contribution is to apply flow-aware dropping to realize fair cycle/s sharing of CPU capacity. Sharing CPU is more complicated since buffering takes place before processing (including selective dropping) and the cycles per packet requirement varies, notably as a function of batch size.

3 FAIR DROPPING

To realize fair dropping we create one virtual scheduler for each of link bandwidth and CPU, in parallel with the actual forwarding logic of the software router. On the arrival of a batch of packets, virtual queues in the form of per-flow counters are decremented by the max-min fair share of service capacity (server rate $\times$ interval) accumulated since the last batch arrival. If the virtual queue of an arriving packet then exceeds a threshold, the packet is dropped. Virtual queues of admitted packets are incremented by the packet size (in bytes for bandwidth, cycles for CPU). Packet size in cycles is estimated by real time measurement of batch processing times.

The algorithm works on the list of flows that are currently *active* in the sense that they have a backlog in the virtual scheduler. This list is highly dynamic with flows being added when they receive a "first" packet and removed when their virtual queue empties. For instance, flows emitting packets at a rate less than the current fair rate enter and rapidly leave the active flow list for each packet arrival. Crucially, under a reasonable stochastic model of flow arrivals, even though the number of flows in progress can attain hundreds of thousands, the number of *active* flows is typically less than 100 for any service rate [4, 5].

4 THE DEMO

To demonstrate feasibility we have implemented the fair dropping algorithm in the Vector Packet Processing (VPP)[1] software router that is part of the Linux Foundation FD.io

[1]https://fd.io/wp-content/uploads/sites/34/2017/07/FDioVPPwhitepaperJuly2017.pdf

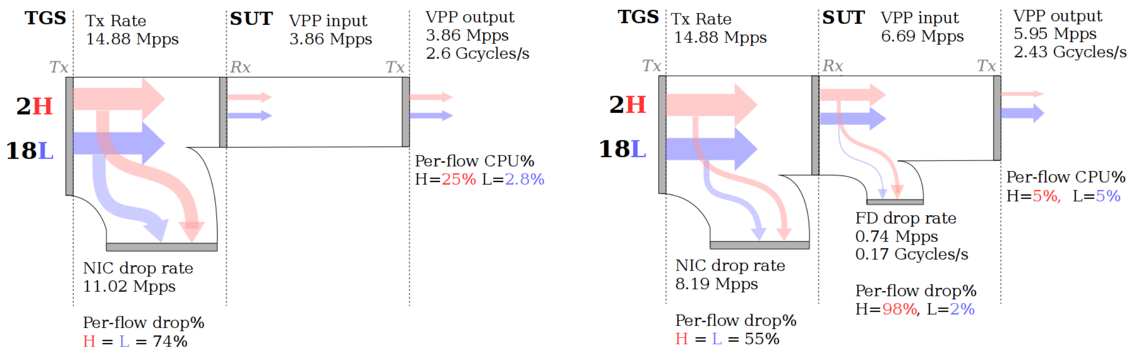
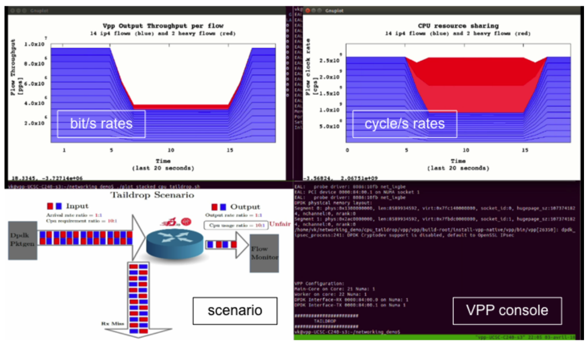


Figure 1: Comparison of tail drop (left) and fair drop (right) for CPU shared between 20 flows: 2 flows (H) require 10 times more cycle/packet that the others (L).



```
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: PCI device 0000:84:00.1 on NUMA socket 1
EAL:   probe driver: 8086:10fb net_ixgbe
DPDK physical memory layout:
Segment 0: phys:0x1300000000, len:8589934592, virt:0x7fc140000000, socket_id:0, hugepage_sz:107374182
4, nchannel:0, nrank:0
Segment 1: phys:0x2ac0000000, len:8589934592, virt:0x7fbdc0000000, socket_id:1, hugepage_sz:107374182
4, nchannel:0, nrank:0
/home/vk/networking_demo/cpu_taildrop/vpp/vpp/build-root/install-vpp-native/vpp/bin/vpp[26350]: dpdk_
ipsec_process:241: DPDK Cryptodev support is disabled, default to OpenSSL IPsec

VPP Configuration:
Main-Core on Core: 21 Numa: 1
Worker on core: 22 Numa: 1
DPDK Interface-RX 0000:84:00.0 on Numa: 1
DPDK Interface-TX 0000:84:00.1 on Numa 1

########################
      TAILDROP
########################
vk@vpp-UCSC-C240-s3:~/networking_demo$
```

Figure 2: Example demo screenshot: CPU shared without fair dropping (https://newnet.telecom-paristech.fr/index.php/fairdrop/)

project. The proposed demo shows how flow rates are modified in two separate scenarios where bandwidth and CPU are separately bottlenecks.

The experimental platform is based on two Intel Xeon E52690 processors, each with 24 cores running at 2.60 GHz, equipped with two 10 Gbps Intel X520 NICs and running DPDK. Two independent nodes are formed, one acting as traffic generator and sink (TGS), the other constituting the system under test (SUT), namely the VPP router modified to perform fair dropping. The TGS continuously sends a stream of packets to the SUT which processes them and sends them back to the TGS for rate monitoring. In the demo, 64-byte packets are sent at 10 Gbps, an input rate of 14.88 Mpps. IP addresses and port numbers are set to emulate a number of distinct constant rate flows. Flows are identified using the 5-tuple hash computed by the NIC and accessible via DPDK `mbuf`. For the demo, the fair dropping algorithms are executed on a single core handling all traffic.

Flow state is stored in a hash-table with 4K rows. An additional data structure identifies the *active* flows enabling data for this set to be maintained in CPU L1 or L2 cache. Timestamps, needed for CPU sharing, are obtained through the DPDK `rxtx_callback` function.

Figure 1 displays results from one experiment. CPU is shared between 2 flows with high (H) cycle/packet requirement and 18 low (L) requirement flows: this Sankey diagram shows how fair dropping reduces buffer overflows while equalizing the cycle/sec rate of all flows.

We have recorded sample videos for both CPU and bandwidth sharing: Figure 2 is a screenshot from the demo and illustrates CPU sharing without fair dropping when 14 L-flows compete with 2 H-flows. The bit/s flow rates are equal but each H-low seizes the same cycle/sec rate as all the L-flows combined. With fair dropping enabled, the cycle/sec rates of L-flows and H-flows are equal.

ACKNOWLEDGMENTS

This work was funded by NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech.

REFERENCES

[1] V. Addanki, L. Leonardo, J. Roberts, and D. Rossi. 2018. Controlling software router resource sharing by fair packet dropping. In *IFIP Networking 2018*.
[2] T. Bonald and J. Roberts. 2015. Multi-Resource Fairness: Objectives, Algorithms and Performance. In *ACM SIGMETRICS*.
[3] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-resource Fair Queueing for Packet Processing. In *ACM SIGCOMM*.
[4] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts. 2005. Evaluating the Number of Active Flows in a Scheduler Realizing Fair Statistical Bandwidth Sharing. In *ACM SIGMETRICS*.
[5] N. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. 2018. Approximating Fair Queueing on Reconfigurable Switches. In *NSDI 18*.
[6] K. To, D. Firestone, G. Varghese, and J. Padhye. 2016. Measurement Based Fair Queuing for Allocating Bandwidth to Virtual Machines. In *ACM HotMiddlebox*.

DEMO REQUIREMENTS

Networking

- The demo relies on Internet access to our server platform located in France. We therefore require a reliable Internet connection.
- The demo runs at10Gbps per line card; however, statistics are aggregated and exported at 1 second granularity so that a low transmission rate is sufficient.

Equipment

A large screen is required for optimal visibility (≥40" screen, ideally). We use our own laptop to pilot the demo.

Space needed

Nothing beyond standard table, screen and poster space.

Set-up time

30 minutes

VIDEOS

For the time being, we have made two videos, available at https://newnet.telecom-paristech.fr/index.php/fairdrop/, that show scenarios where the system bottleneck is CPU and bandwidth, respectively. The following describes the CPU video.

The video at https://perso.telecom-paristech.fr/drossi/data/videos/fairdrop-cpu-demo.mp4 presents a recorded demo for CPU sharing. This and the similar video showing bandwidth sharing with and without fair dropping will be used as backup in case of technical problems. The video has no soundtrack and needs commentary by the presenter. Its timeline is as follows:

time	frame	comment
0 sec	scenarios to be compared	red packets require 10× more cycles per packet than blue
8 sec	traffic generator is launched (top left)	DPDK Pktgen
17 sec	flow monitor is launched (top right)	
26 sec	unmodified VPP is launched (bottom right)	
40 sec	real time plots of flow rates appear	packet/s (top left) and cycle/s (top right)
		2 red flows and 14 blue flows start together
		red flows run for 10 s, stop and start again after 10s
		flows have equal packet rates while red flows
		grab 10 times more CPU than blue flows
80 sec	modified VPP with fair dropping launched	
100 sec	plots of flow rates appear	flows have equal cycle/s rates as red packets are
		dropped aggressively
140 sec	the end!	