Fair dropping for multi-resource fairness in software routers

Extended Abstract

Vamsi Addanki, Leonardo Linguaglossa, James Roberts, Dario Rossi

Telecom ParisTech

Paris, France

firstname.lastname@telecom-paristech.fr

ABSTRACT

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 elsewhere [1].

CCS CONCEPTS

• **Networks** → **Packet scheduling**; **Packet-switching networks**; *Transport protocols*;

KEYWORDS

multi-resource fairness, fair dropping, vector packet processing (VPP), Linux Foundation FD.io project

ACM Reference Format:

Vamsi Addanki, Leonardo Linguaglossa, James Roberts, Dario Rossi. 2018. Fair dropping for multi-resource fairness in software routers: Extended Abstract. In *Posters-Demos '18: Posters and Demos, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3234200.3234210>

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Posters-Demos '18, August 20–25, 2018, Budapest, Hungary

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5915-3/18/08...\$15.00

<https://doi.org/10.1145/3234200.3234210>

applications might have different tolerance for packet latency and loss.

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.* [4] 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 [4], 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 [8]. 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* (e.g., [6]). Our approach takes account of the specifics of the software router and is more precise than 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 [5, 7].

4 THE DEMO

To demonstrate feasibility we have implemented the fair dropping algorithm in the Vector Packet Processing (VPP) software router that is part of the Linux Foundation FD.io project [3]. The demo shows how flow rates are modified in two 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. Traffic is generated from the TX port using the standard pktgen-DPDK [9], while flow rates are measured in the RX port with FlowMon-DPDK [10]. 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

Figure 1: Comparison of tail drop and fair drop for CPU shared between 20 flows: 2 flows (H) require 10 times more cycles per packet than the others (L).

are identified using the 5-tuple hash computed by the NIC and accessible via the DPDK mbuf. For the demo, the fair dropping algorithms are executed on a single core handling all traffic though we have also implemented a multicore scenario.

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.

ACKNOWLEDGMENTS

This work was funded by NewNet@Paris, Cisco’s Chair “NETWORKS FOR THE FUTURE” at Telecom ParisTech and performed at LINC (http://www.lincs.fr).

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] FD.io. 2016. VPP whitepaper. <https://fd.io/wp-content/uploads/sites/34/2017/07/FDioVPPwhitepaperJuly2017.pdf>. (2016).
- [4] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-resource Fair Queueing for Packet Processing. In *ACM SIGCOMM*.
- [5] A. Kortebe, 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*.
- [6] Rong Pan, Lee Breslau, Balaji Prabhakar, and Scott Shenker. 2003. Approximate Fairness Through Differential Dropping. *ACM SIGCOMM Comput. Commun. Rev.* 33, 2 (April 2003), 23–39. <https://doi.org/10.1145/956981.956985>
- [7] N. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. 2018. Approximating Fair Queueing on Reconfigurable Switches. In *NSDI 18*.
- [8] K. To, D. Firestone, G. Varghese, and J. Padhye. 2016. Measurement Based Fair Queueing for Allocating Bandwidth to Virtual Machines. In *ACM HotMiddlebox*.
- [9] Intel DPDK traffic generator. 2016. <http://git.dpdk.org/apps/pktgen-dpdk/tag/?h=pktgen-3.1.2>. (2016).
- [10] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. 2018. FlowMon-DPDK: Parsimonious per-flow software monitoring flatline rate. In *IFIP Network Traffic Measurement and Analysis Conference (TMA)*.