A Distributed Algorithm to Calculate Max-Min Fair Rates Without Per-Flow State

Lavanya Jose
lavanyaj@cs.stanford.edu
Stanford University
Stanford, CA, USA

Stephen Ibanez
sibanez@stanford.edu
Stanford University
Stanford, CA, USA

Mohammad Alizadeh
alizadeh@csail.mit.edu
MIT CSAIL
MA, USA

Nick McKeown
nickm@stanford.edu
Stanford University
Stanford, CA, USA

ABSTRACT

Most congestion control algorithms, like TCP, rely on a reactive control system that detects congestion, then marches carefully towards a desired operating point (e.g. by modifying the window size or adjusting a rate). In an effort to balance stability and convergence speed, they often take hundreds of RTTs to converge; an increasing problem as networks get faster, with less time to react.

This paper is about an alternative class of congestion control algorithms based on proactive-scheduling: switches and NICs "proactively" exchange control messages to run a *distributed* algorithm to pick "explicit rates" for each flow. We call these Proactive Explicit Rate Control (PERC) algorithms. They take as input the routing matrix and link speeds, but not a congestion signal. By exploiting information such as the number of flows at a link, they can converge an order of magnitude faster than reactive algorithms.

Our main contributions are (1) s-PERC ("stateless" PERC), a new practical distributed PERC algorithm without per-flow state at the switches, and (2) a proof that s-PERC computes exact max-min fair rates in a known bounded time, the first such algorithm to do so without per-flow state. To analyze s-PERC, we introduce a parallel variant of standard waterfilling, 2-Waterfilling. We prove that s-PERC converges to max-min fair in $6N$ rounds, where N is the number of iterations 2-Waterfilling takes for the same routing matrix.

We describe how to make s-PERC practical and robust to deploy in real networks. We confirm using realistic simulations and an FPGA hardware testbed that s-PERC converges 10-100x faster than reactive algorithms like TCP, DCTCP and RCP in data-center networks and 1.3–6x faster in wide-area networks (WANs). Long flows complete in close to the ideal time, while short-lived flows are prioritized, making it appropriate for data-centers and WANs.

KEYWORDS

congestion control; max-min fairness; data center networks

SIGMETRICS'19 Abstracts, June 24–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6678-6/19/06.
https://doi.org/10.1145/3309697.3331472

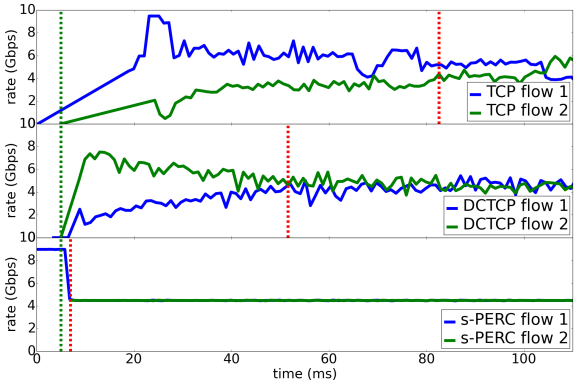

Figure 1: Typical convergence behavior for TCP, DCTCP, and s-PERC running on our NetFPGA testbed.

ACM Reference Format:
Lavanya Jose, Stephen Ibanez, Mohammad Alizadeh, and Nick McKeown. 2019. A Distributed Algorithm to Calculate Max-Min Fair Rates Without Per-Flow State. In *ACM SIGMETRICS Int'l Conference on Measurement & Modeling of Computer Systems (SIGMETRICS'19 Abstracts), June 24–28, 2019, Phoenix, AZ, USA*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3309697.3331472

EXTENDED ABSTRACT

Cloud data-centers today host thousands of applications on networks interconnecting hundreds of thousands of servers. A congestion control algorithm has to balance the needs of short flows, which need low latency, and long flows, which need high throughput, regardless of other applications sharing the network.

Cloud providers typically use reactive congestion-control algorithms inside and between their data-centers. This class of algorithms, best represented by TCP, *reacts* to congestion signals, and can take hundreds of RTTs to converge, even for simple topologies. Figure 1 illustrates the problem in our hardware testbed. Two servers send a TCP flow over a 10Gb/s bottleneck link. The round-trip time (RTT) is 1ms. After the second flow is added (first vertical line), it takes TCP 400 RTTs to converge to a fair allocation (second vertical line). Even DCTCP takes 250 RTTs.

In real networks, many flows are short-lived and the set of flows changes every millisecond, if not more often, suggesting that instantaneous flow rates in today's networks never have time to

converge and are far from optimal. As link speeds increase this problem becomes worse, since flows can finish even faster.

An alternative would be to use scheduling algorithms like WFQ [2] or PGPS [5], which instantaneously share link capacity fairly across all flows using a link. But maintaining per-flow state is too expensive in data-center switch ASICs with limited on-chip memory. Another approach, taken by FlowTune [6], is to calculate a fair rate allocation for each flow in a centralized server, and schedule the flows to be sent at these rates. But a centralized scheduler is a bottleneck in a large network, with a rapidly changing set of flows.

We seek fast congestion control algorithms that are practical, do not require per-flow state, allowing them to scale to any number of flows, and are distributed, allowing them to scale to any network size. They must converge quickly to fair rates for any number of flows. They must remain stable and fast in the face of sudden changes in the traffic matrix. Finally, they must be general enough to work for arbitrary topologies at WAN or DC scale.

Our approach is to use distributed *proactive* (rather than reactive) algorithms to directly calculate the ideal flow rates. We focus on a class of congestion control algorithms where switches and NICs "proactively" exchange messages (control packets) to run a *distributed* algorithm to pick "explicit rates" for each flow. We call this Proactive Explicit Rate Control (PERC) [4]. The end hosts send and receive control packets for each flow, and switches on the flow's path use these control packets to locally calculate a rate to allocate to the flow. The source end host of a flow updates the flow's sending rate each time it receives a control packet, to match the lowest rate allocated by any switch. Over multiple round trips of the control packets, PERC algorithms figure out the ideal rates exactly for every flow. PERC algorithms take as input the routing matrix and link speeds, but not a congestion signal. By exploiting information such as the number of flows at a link, they can converge an order of magnitude faster than reactive algorithms.

Introducing a PERC congestion control algorithm into a network presents many challenges. But as we have seen in recent years, cloud providers seem willing to invest the effort, given their homogeneous infrastructure and single administrative domain. Recent programmable switches make it practical to implement simple distributed algorithms at switches, that collect information about flows proactively and act on it quickly.

Contributions: Our main contributions are (1) s-PERC ("stateless" PERC), a new practical distributed PERC algorithm without per-flow state at the switches, and (2) a proof that s-PERC computes exact max-min fair rates in a known bounded time, the first such algorithm to do so without per-flow state.

The s-PERC algorithm is a deceptively simple distributed algorithm: End hosts send and receive control packets that carry four fields (< 7B total) per link, and switches use these control packets to locally calculate the exact max-min fair flow rate, using a constant amount of state (8B per link) at the switch itself. s-PERC was designed to work with the max-min fair metric because it is a widely-used objective for congestion control algorithms. We describe how to make s-PERC practical and robust for DC and WAN networks, including how to handle lost control packets. We have built a hardware prototype of s-PERC using the 40Gb/s NetFPGA SUME platform.

Convergence Result: To get a close to optimal bound on the convergence time of s-PERC, we introduce a family of *centralized* algorithms called k-Waterfilling algorithms [3]. The well-known sequential water-filling algorithm [1] is the special case when $k = \infty$. As we make k smaller, the water-filling algorithm becomes more parallel and needs fewer iterations to compute max-min fair rates. To set the stage for s-PERC, we review an existing distributed PERC algorithm, called *Fair*, which requires per-flow state at the links, and computes locally max-min fair rates at every link to converge to a global max-min fair allocation [7]. Previous work shows that the convergence behavior of *Fair* can be analyzed using 1-Waterfilling [7], the fastest and most parallel of our water-filling algorithms. We show that the convergence behavior of s-PERC can be analyzed using 2-Waterfilling, the second most parallel water-filling algorithm. Specifically, we show that *s-PERC is guaranteed to converge to the max-min fair allocation within* $6N$ *rounds, where* N *is the number of iterations* 2*-Waterfilling takes for the same routing matrix and link speeds, and round is the maximum round trip time of all control packets [3].* This is a tighter bound than we can obtain using the standard water-filling algorithm ($k = \infty$), but looser than the $k = 1$ bound for *Fair*. The intuition is that because *Fair* uses more state (specifically, per-flow state at every link) it can be made more parallel and therefore converges faster. But s-PERC also runs in parallel and can converge much faster than standard water-filling, despite requiring no per-flow state at the links.

Practical Evaluation: Numerical simulations using randomly generated routing matrices indicate that s-PERC converges 10-40% slower than *Fair* in practice, and the worst-case convergence time for s-PERC was 2-3x faster than the $6N$ bound. Packet-level simulations with realistic workloads show that s-PERC converges an order of magnitude faster than existing reactive algorithms in data center networks, and achieves close to ideal throughput for large flows, and near-minimum latency for the smallest flows. In WAN networks, s-PERC converges 1.3–6x faster. Testbed measurements show that s-PERC converges two orders of magnitude faster than TCP and DCTCP in simple topologies.

REFERENCES

[1] Dimitri Bertsekas and Robert Gallager. 1987. *Data Networks.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[2] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. In *Symposium Proceedings on Communications Architectures &Amp; Protocols (SIGCOMM '89).* ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/75246.75248

[3] Lavanya Jose, Stephen Ibanez, Mohammad Alizadeh, and Nick McKeown. 2019. A Distributed Algorithm to Calculate Max-Min Fair Rates Without Per-Flow State. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 2, Article 21 (June 2019), 42 pages. https://doi.org/10.1145/3326135

[4] Lavanya Jose, Lisa Yan, Mohammad Alizadeh, George Varghese, Nick McKeown, and Sachin Katti. 2015. High Speed Networks Need Proactive Congestion Control. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets-XIV).* ACM, New York, NY, USA, Article 14, 7 pages. https://doi.org/10.1145/2834050.2834096

[5] Abhay K Parekh and Robert G Gallager. 1993. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking* 1, 3 (1993), 344–357.

[6] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. 2017. Flowtune: Flowlet Control for Datacenter Networks. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17).* USENIX Association, Berkeley, CA, USA, 421–435. http://dl.acm.org/citation.cfm?id=3154630.3154665

[7] Jordi Ros-Giralt. 2003. *A Theory of Lexicographic Optimization for Computer Networks.* Ph.D. Dissertation. University of California, Irvine.