Research Impacting the Practice of Congestion Control

Nandita Dukkipati, Yuchung Cheng, Amin Vahdat

Google Inc.

{nanditad, ycheng, vahdat}@google.com

1. INTRODUCTION

Many algorithms proposed in networking research papers are widely used in many areas, including Congestion Control, Routing, Traffic Engineering, and Load Balancing. In this paper, we present algorithmic advancements that have impacted the practice of Congestion Control (CC) in datacenters and the Internet. Where possible, we also describe negative examples, ideas that looked promising on paper or in simulations but that performed poorly in practice. We conclude the paper with observations on the characteristics shared by these ideas in taking them from research to impacting practice.

2. DATACENTER CONGESTION CONTROL

There has been substantial innovation in datacenter networking for a number of reasons. First, the workloads have novel requirements requiring both low latency and high bandwidth. Hence, networking improvements have substantial impact on end application performance. Second, datacenters are tightly controlled regimes where end-hosts and switch changes can be deployed in a coordinated manner with less absolute concern over backward compatibility, incremental deployment and fairness to legacy protocols.

There are two classes of congestion control algorithms in the large TCP literature that attempt to control latency while trying to sustain large throughput: 1) Active queue management approaches use explicit feedback such as ECN from congested switches; 2) Delay-based algorithms use the increase in round-trip time (RTT) measurements above a baseline propagation delay as an indication of queueing delay and congestion. Algorithms in both of these classes have impacted the practice of datacenter CC.

Datacenter TCP (DCTCP) [1] made ECN based CC the defacto standard in datacenters. In DCTCP, a simple active queue management scheme uses a single parameter, the queue occupancy threshold, to specify the threshold for marking packets with the Congestion Experienced codepoint. The single threshold ensures that sources are quickly notified of the queue overshoot. End hosts extract multibit feedback from the single bit stream of ECN marks. Sources use the fraction of marked packets as an estimate for the extent of network congestion. Experiments with real datacenter workloads showed DCTCP can operate with low buffer occupancy while still achieving high throughput.

TCP Vegas [2] is a pioneering work in first targeting to achieve high bandwidth *and* low network delay. Its key insight is to use end host measurements of packet RTT as a signal to incipient congestion. Until recently, a delay based CC was considered hard for datacenters because measuring round-trip time (RTT) is susceptible to noise in low-latency environments. Small noisy fluctuations of delay can easily become indistinguishable from delay variations due to congestion. Using recent advances of NIC hardware to accurately measure RTTs and building upon the seminal work of TCP Vegas, delay based CC such as TIMELY [12] have impacted the practice of congestion control for RDMA networks. TIMELY demonstrated that accurate RTT measurements provide reliable estimates of microsecond timescale increases of end-to-end queueing latency. Using congestion control similar to that of TCP Vegas, TIMELY bounds the 99th percentile tail latency of delay sensitive datacenter applications.

The final algorithm we want to highlight in datacenters is Water Filling algorithms. Their use in networking systems to achieve max-min fair rate allocation for a general topology is described well by D. Bertsekas and R. Gallager [5]: the key idea is to start with an all-zero rate vector and to increase the rates on all paths together until one or more links is saturated. At this point, each session using a saturated link has the same rate as all other sessions using that link. The saturated links serve as bottleneck links for all sessions using them. In subsequent steps of the algorithm, all sessions not using the saturated links have their allocations incremented equally until one or more new links become saturated. The algorithm continues in every step to equally increment rate of all sessions which are not passing through any saturated link. The algorithm stops when all sessions pass through at least one saturated link. Water Filling algorithms are used in a variety of networking systems for providing isolation amongst different entities. A good example is the Bandwidth Enforcer (BwE) [10] system at Google that distributes bandwidth from users to jobs, and from jobs to tasks using Water Filling to provide for max-min fair allocation.

3. INTERNET CONGESTION CONTROL

After a flurry of research in early 2000s on scaling Internet CC to high bandwidth-delay product networks, CUBIC TCP [8] emerged as the default CC in Linux. CUBIC extends the classic Reno CC. Instead of fixed additive increment, its window growth is a cubic function relative to the elapsed time since the last loss event. The window grows quickly upon a window reduction, but as it gets closer to the last window experiencing packet loss, it slows its growth (the window increment becomes almost zero). Above that, CUBIC starts probing for more bandwidth where the window grows slowly initially, and accelerates its growth as it moves away from the last loss window. This slow growth around loss windows enhances the algorithm stability, while the fast growth away from the loss window lets CUBIC scale to higher bandwidth. One interesting note is that rapid evolution in Linux CC was facilitated by the introduction of Linux CC modularization enabling A/B comparisons of CC algorithms with no kernel recompilation.

Nevertheless CUBIC still scales poorly with increasing WAN capacity because of the impractically low loss rates required to deliver Gbps rates. Losses over the last decade are burstier both on

the Internet and across data centers due to a combination of batch-and-burst techniques (e.g., offload techniques), the advent of small buffers on commodity switches, and token-bucket rate policing.

The direct approach to reduce burstiness is to increase packet mixing and to maximize traffic entropy. The classic technique of Fair Queueing [6] [13] achieves these goals very well. A centralized global FQ scheduler for all the flows can support software pacing where the rate is set on a per TCP flow basis. FQ/pacing [4] queuing discipline (Qdisc) works as follows: a TCP flow injects packets into the Qdisc. The FQ scheduler round-robins all the active flows with a quantum of two MTU packets. A packet is sent immediately if the flow has enough credit, otherwise it is scheduled at a release time based on the packet size divided by the flow's pacing rate. Pacing rate is set using the flow's congestion window and RTT. FQ/pacing reduced bursts more effectively than pure pacing because sending gaps created at the originating hosts are often eliminated toward the destination that eventually result in bursts. Instead FQ replaces the gaps by packets from other flows at the originating host. As we rolled out FQ/pacing at Google we saw distinct utilization increase and loss reduction. In particular, many losses on our peering links moved to the destination access links. Prior to FQ/pacing deployment, our peering routers dropped large TSO bursts. Therefore, CUBIC delivered poor performance by reacting to burst losses at the peering points early instead of creating queues later at the access links. The deployment eliminated such losses and allowed CUBIC to probe for the real bottleneck, typically at the access links.

While FQ/pacing reduces losses, it does not eliminate them. Recovering losses quickly is still key to performance. For example, TCP Forward Acknowledgments (FACK) [11] is a loss detection algorithm in TCP using Selective Acknowledgements (SACK). The key insight is the notion of using time sequence in detecting losses: if packet A was sent after packet B, and if the acknowledgement for packet A is received first, then FACK infers that packet B must be lost. FACK is implemented in Linux in early 2000 and has had more impact beyond that. Subsequent loss recovery (e.g., Tail Loss Probes [7], RACK [3]) and other protocols (e.g, QUIC [9]) have embraced the concepts FACK introduced. The success of FACK shows that simple but insightful ideas can influence the global Internet and generations of transport stacks. Another invention is the *pipe* algorithm that estimates the number of packets in flight by counting the packets delivered out-of-order, retransmitted, and lost. The pipe algorithm simplifies the TCP stack by decoupling loss recovery and CC.

Caching TCP parameters based on IP addresses [14] turned out to be less impactful than we anticipated. The idea assumes both spatial and temporal locality of network properties on the IP path. Therefore caching network or TCP parameters (e.g., RTT, initial window etc.) should improve performance. However, in practice, this approach can be more harmful than helpful. First the prevalence of carrier-grade NAT and DHCP means that the same IP prefix does not imply the same network path. Second, the number of active connections on a network path is highly dynamic, e.g., the cache may remember a low slow start threshold for a short connection that suffered an RTO due to burst losses. A new bulk connection on the same IP prefix will exit slow start early and receive poor throughput. These factors make caching TCP state information ineffective. Nevertheless an open-source implementation still helps us evaluate these research proposals more conveniently. A potential research direction is to apply machine learning to identify meaningful network signals amenable to caching.

4. FROM RESEARCH TO PRACTICE

There are many paths for research to impact the practice of congestion control. In our experience, the paths have been different for datacenter CC relative to Internet CC. Datacenter environments offer the luxury of a single administrative domain, and hence issues such as backwards compatibility are less of a concern. Instead, the two key factors that we have observed for success in DC environment are: 1) simplicity and the effectiveness of the schemes in solving a real problem, and 2) champion influencer(s) to *make it happen*. The influencers could be anyone from engineers to management in authority. These same factors apply for Internet CC as well, except the influencer or the champion either needs to partner with or be a part of the open source community, which in our specific examples is the Linux OS community.

5. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM '10*.

[2] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM '94*.

[3] Y. Cheng and N. Cardwell. RACK: a time-based fast loss detection algorithm for TCP, 2016. https://tools.ietf.org/html/draft-cheng-tcpm-rack-00.

[4] J. Corbet. Tso sizing and the fq scheduler. https://lwn.net/Articles/564978/.

[5] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall International Editions, 1987.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*

[7] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis. Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses, 2013. https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01.

[8] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Operating System Review '08*.

[9] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2, 2016.

[10] A. Kumar, S. Jain, U. Naik, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Sigcomm '15*, 2015.

[11] M. Mathis and J. Mahdavi. Forward acknowledgment: Refining tcp congestion control. In *In Proceedings of the ACM SIGCOMM*, 1996.

[12] R. Mittal, T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. Timely: Rtt-based congestion control for the datacenter. In *Sigcomm '15*, 2015.

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

[14] J. Touch. TCP Control Block Interdependence. RFC 2140, 1997.