Data Center TCP - Reproducing Key Findings

CS 656 Research Paper

Colin Howes
University of Waterloo
200 University Ave. W
Waterloo, ON N2L 3G1
chowes@uwaterloo.ca

ABSTRACT

Data centers supporting modern large-scale applications must host a diverse range of heterogeneous traffic with varying throughput and delay requirements. Transport layer protocols facilitating data center traffic must be able to tolerate intense bursts of traffic, provide low latencies for time-sensitive traffic, all while maintaining high throughput for large data flows. TCP makes up the majority of traffic in modern data centers, though a number of performance impairments suggest that traditional TCP congestion control mechanisms are ill-suited to the demands of a data center environment. Several modifications to the TCP protocol have been proposed to address shortcomings in conventional TCP congestion control algorithms, which relies on packet loss as a metric of network congestion. This paper presents an overview of traffic problems in modern data centers, as well as a number of proposed modifications to TCP's congestion control mechanisms aimed at improving the performance of TCP for typical data center communication patterns. Finally, key results related to the performance of Data Center TCP are replicated and discussed.

1 INTRODUCTION

Data centers are responsible for driving large-scale web applications such as web search, storage, advertising, and social network composition [1, 4]. These applications generate diverse traffic patterns with strict throughput and latency requirements. In particular, many distributed web applications rely on a workflow patterns in which requests are broken down and distributed to multiple workers, which perform tasks simultaneously and return responses to an aggregator [1, 4]. As a result, data centers need to support bursts of many-to-one traffic traversing shared bottlenecks without affecting the throughput of long-lived flows needed to update and maintain application data [1].

In order to accommodate the workload generated by these applications while maintaining cost efficiency, modern data centers typically feature high speed links with very low propagation delays connecting nodes via low-cost switches with shallow buffers [1, 4, 7]. The majority of communication between nodes is over TCP, which was designed based on the characteristics of wide area networks, where round-trip times (RTTs) are orders of magnitude greater than in data centers [4]. As a result, while traditional TCP congestion control mechanisms which are usually considered "good enough" in the context of the internet, TCP does not perform well when faced with traffic patterns typical of data center networks [4, 8].

In particular, the traffic patterns common to data center networks suffer a number of impairments due to problems with traditional TCP congestion control mechanisms. This paper discusses a number of approaches to TCP congestion control aimed at improving the performance of TCP in data centers. In particular, Data Center TCP (DCTCP) is discussed in depth, and selected key findings are reproduced using the Mininet network emulator.

2 DATA CENTER TRAFFIC PATTERNS

Distributed applications running in modern data centers are based on the *partition/aggregate* design pattern, in which an application is broken in into hierarchical layers and time-sensitive requests at higher layers are divided and delegated to workers in the lower layers. Many workers perform a small component of a task and return a response to an aggregator, which is combined with responses from other workers and passed back up through the hierarchy. An aggregator combines these responses to get a result which can then be sent back to the user. This model is the basis of many data processing frameworks, cluster storage systems, and targeted advertising applications [1, 4, 5, 8]. These applications require extremely low latencies for short query traffic between workers and aggregators, as well as high throughput for long background flows needed to maintain the freshness of application data and update control state at worker nodes [1].

2.1 Performance Impairments

The current state of TCP congestion control underlies a number of problems that make meeting the demands of distributed applications based on the partition/aggregate pattern difficult to accomplish.

2.1.1 Incast. TCP throughput collapse, or *incast congestion*, arises in many-to-one communication patterns where messages from multiple senders to a single receiver are synchronized and must traverse a shared bottleneck. In incast scenarios, throughput at the receiver collapses as the number of concurrent senders increases and incoming packets overwhelm switch buffers, leading to packet loss and high queuing delays [4, 8]. Incast congestion emerges in the partition/aggregate pattern commonly used in distributed applications, as responses from workers to a single aggregator must cross a shared bottleneck and tend to be synchronized [1].

2.1.2 Buffer Bloat and Underflow. The "additive increase, multiplicative decrease" approach to congestion control in traditional TCP implementations causes long flows to build up buffer queue length until a buffer is overwhelmed and packet loss is detected, at which point the TCP send window is cut, and the sender once again begins to slowly increase the send window. In effect, this approach to congestion control leads to queue buildup, or buffer bloat, which causes long queueing delays even in the absense of packet loss. Conversely, when buffers are small, packet loss is mistakenly interpreted as network congestion, leading to underutilization of network resources [3].

In data centers, this causes latency critical query traffic to incur queuing delays in the presence of long-lived background flows, which steadily build up and maintain large buffer queues. Moreover, improving resource utilization through dynamic buffer allocation means that long lived flows can also cause buffer pressure that impacts flows traversing different ports on the same switch [1].

3 IMPROVING TCP FOR DATA CENTERS

As a result of the performance impairments discussed above, developers have adopted provisional solutions that incur performance costs that impact the overall usefulness of the application. For example, incast congestion can be avoided by adding random delays to query traffic, which reduces the overall response time at higher levels of the hierarchy [1, 6]. The solutions presented here attempt to address issues underlying incast congestion and buffer utilization problems by overhauling TCP's congestion control mechanisms and making use of multiple paths available in highly connected data center networks.

3.1 Incast TCP

3.2 Multipath TCP

3.3 TCP BBR

TCP BBR (Bottleneck, Bandwidth, and Round-trip propagation time) uses "congestion-based congestion control" in place of traditional loss-based congestion detection mechanisms to improve the performance of TCP. TCP BBR is intended as a general replacement for traditional TCP congestion control mechanisms, though it is currently being implemented in a number of Google services including the Google Cloud Platform as well as Google's B4 Backbone, which is a wide area network comprised of shallow buffered commodity switches similar to those used in data centers [2, 3].

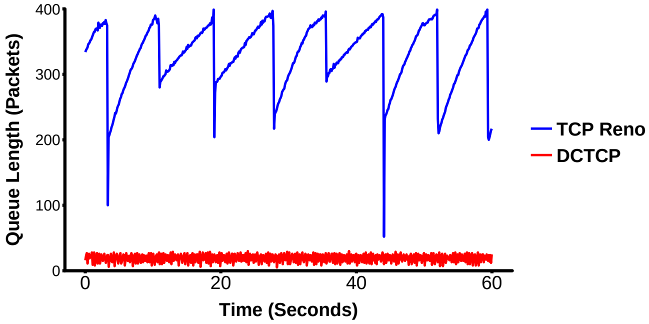

Figure 1: Comparison of queue length over time between DCTCP and TCP Reno with 2 flows

3.4 Data Center TCP

Data center TCP (DCTCP) attempts to address the problem of latency in partition/aggregate traffic by reducing queue length without affecting throughput for large TCP flows.

4 REPRODUCING KEY DCTCP RESULTS

4.1 Methods and Results

Selected results from [1] were reproduced using the Mininet network emulator running on Ubuntu 12.04 with a version of the 3.2.18 Linux kernel patched to add in support for DCTCP. While the original study used link speeds of 1 Gbps and 10 Gbps, initial tests showed Mininet was unable to maintain consistent throughput above 100 Mbps even when running on a compute optimized EC2 instance with support for high bandwidth networking (see Limitations). Thus, a link speed of 100 Mbps was used in all tests in order to acheive consistent results.

4.1.1 Queue Length. In order to replicate the queue length results shown in Figures 1 and 13 [1], a simple topology of N flows was constructed with N senders continuously transferring data to a single receiver over a shared bottleneck with a maximum queue length of 400 packets. Figure 1 shows queue length over time for two sustained flows over the shared bottleneck. DCTCP maintained a small steady queue length ($M = 19.38$), while TCP New Reno kept the buffers consistently full with much greater variance in queue length ($M = 330.31$), and exhibited the characteristic sawtooth pattern, with the buffer reaching maximum occupancy at 400 packets before dropping when a loss was detected. Furthermore, Figure 2 shows that DCTCP was much less sensitive to N, and maintained consistently low queue sizes at both $N = 2$ and $N = 20$ ($M = 23.63$), whereas queue length for TCP New Reno was consistently greater at $N = 20$ ($M = 368.99$) than when $N = 2$.

4.1.2 Throughput. Both DCTCP and TCP New Reno made nearly full use of the 100 Mbps bandwidth, maintaining an average of 98.56 and 98.39 Mbps throughput respectively

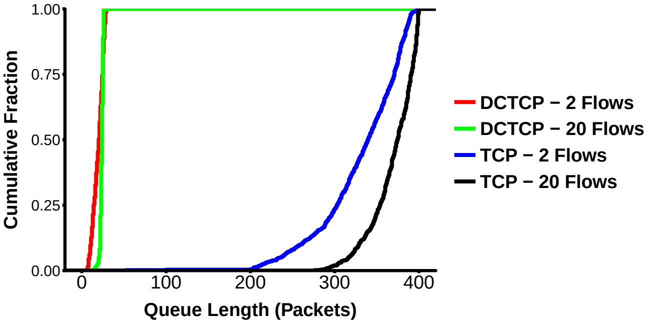

Figure 2: CDF of queue length for DCTCP and TCP Reno with 2 and 20 flows

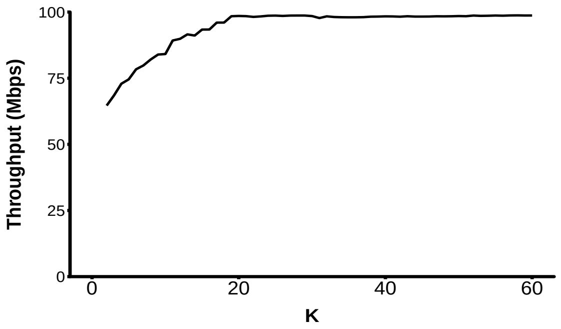

Figure 3: Throughput as a function of K

for $N = 2$, which is consistent with the results of the original paper [1].

The model presented by the authors suggests setting K as a function of link speed and RTT, and an initial test with an RTT of 100μs and a link speed of 100 Mbps showed that throughput was insensitive to K, which is consistent the authors' findings in tests with link speeds of 1 Gbps. Therefore, in order to compensate for limited bandwidth, the RTT was increased to 1 ms and the test was repeated. Figure 3 shows the total throughput of two flows transferring data to a single receiver as a function of the marking threshold, K. The results under these conditions showed that throughput increased with K until the maximum throughput was reached at $K \approx 20$.

4.1.3 Convergence. Convergence was tested by incrementally starting 5 flows transmitting to a single receiver across a shared bottleneck and allowing each to run for 60 seconds before adding the next flow. Once the fifth flow had run for 60 seconds, each of the 5 flows was stopped in 60 second intervals. The results of the DCTCP convergence test (Figure 4) show that each of the DCTCP flows quickly converges to its fair share of the available bandwidth with little variability. In addition, the total bandwidth across the flows indicated consistently high throughput as flows were started and stopped.

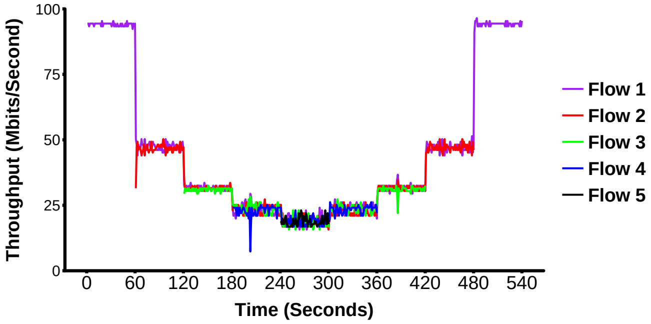

Figure 4: Convergence of 5 flows DCTCP

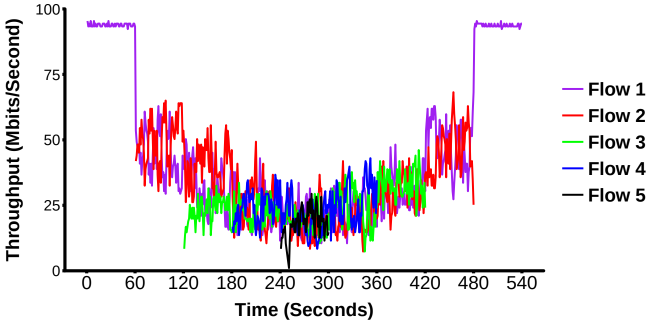

Figure 5: Convergence of 5 flows TCP Reno

Interestingly, the results of the TCP New Reno convergence test showed considerably more variability.

4.2 Limitations

5 CONCLUSIONS

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, Vol. 40. ACM, 63–74. http://dl.acm.org/citation.cfm?id=1851192

[2] Neal Cardwell. 2017. TCP BBR congestion control comes to GCP your Internet just got faster. (2017). https://cloudplatform.googleblog.com/2017/07/TCP-BBR-congestion-control-comes-to-GCP-your-Internet-just-got-faster.html

[3] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5 (Oct. 2016), 50:20–50:53. https://doi.org/10.1145/3012426.3022184

[4] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. 2009. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/1592681.1592693

[5] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. *Com-munications ofthe ACM* 51, 1 (2004), 107–l.

[6] S. Floyd and V. Jacobson. 1994. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking* 2, 2 (April 1994), 122–136. https://doi.org/10.1109/90.298431

[7] James Hamilton. 2007. On Designing and Deploying Internet-Scale Services. In *Proceedings of the 21st Large Installation System Administration Conference*. Austin, TX, 231–242.

[8] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson, and Srinivasan Seshan. 2008. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems.. In *FAST*, Vol. 8. 1–14.