Bullet Trains: A Study of NIC Burst Behavior at Microsecond Timescales

Rishi Kapoor, Alex C. Snoeren, Geoffrey M. Voelker, George Porter
University of California, San Diego

Abstract

While numerous studies have examined the macro-level behavior of traffic in data center networks—overall flow sizes, destination variability, and TCP burstiness—little information is available on the behavior of data center traffic at packet-level timescales. Whereas one might assume that flows from different applications fairly share available link bandwidth, and that packets within a single flow are uniformly paced, the reality is more complex. To meet increasingly high link rates of 10 Gbps and beyond, batching is typically introduced across the network stack—at the application, middleware, OS, transport, and NIC layers. This batching results in short-term packet bursts, which have implications for the design and performance requirements of packet processing devices along the path, including middleboxes, SDN-enabled switches, and virtual machine hypervisors.

In this paper, we study the burst behavior of traffic emanating from a 10-Gbps end host across a variety of data center applications. We find that at 10–100 microsecond timescales, the traffic exhibits large bursts (i.e., 10s of packets in length). We further find that this level of burstiness is largely outside of application control, and independent of the high-level behavior of applications.

1. INTRODUCTION

As an increasing array of sophisticated applications move to the cloud, Internet data centers must adapt to meet their needs. Unlike software installed on a single machine, data center applications are spread across potentially thousands of hosts, resulting in increasingly stringent requirements placed on the underlying network interconnect. Data center networks must isolate different tenants and applications from each other, provide low-latency access to data spread across the cluster, and deliver high bisection bandwidth, all despite omnipresent component failures.

Administrators deploy a variety of packet processing technologies to achieve these goals, spanning all layers of the network stack. Virtual machine hypervisors implement complex virtual switched networks [18], ferrying packets between VMs and the rest of the data center. Individual data flows are forwarded through a variety of middleboxes [23], which process data packets to implement a variety of value-added services. Most recently, the adoption of software-defined networking (SDN) means that software controllers are involved in providing basic connectivity services within the network—a task previously assigned to custom, high-performance hardware devices [17]. The result of each of these trends is that more devices, running a mixture of software and hardware, sit on the data plane, handling packets. At the same time, link rates continue to increase, first from 1 Gbps to 10 Gbps, and now 10 Gbps to 40 Gbps, 100 Gbps [1], and beyond. Given the increasing speed of network links, and increasing complexity of packet processing tasks, handling these data flows is a growing challenge [23].

Designing efficient and performant packet processing devices, either in software or hardware, requires a deep understanding of the traffic that will transit them. Each of these devices has to carefully manage internal resources, including TCAM entries, flow entry caches, and CPU resources, all of which are affected by the arrival pattern of packets. In their seminal work, Jain and Routhier studied the behavior of traffic transiting Ethernet networks at a packet-level granularity [12], and found that the traffic exhibited a number of unexpected behaviors, including the formation of "trains" which are sets of packets, closely grouped in time, from a single source to a particular destination. Since this analysis was undertaken, networks have evolved considerably. For example, instead of 10-Mbps shared-medium Ethernet, modern data center networks rely on 10-Gbps switched Ethernet.

In this work, we analyze the output of 10-Gbps servers running a variety of data center workloads. We define a *burst* to be an uninterrupted sequential stream of packets from one source to one destination with an inter-packet spacing of less than 40 ns, which is four times the interframe gap (IFG) for 10-Gbps Ethernet. We find that the traffic exhibits bursty "train" type behavior, similar to Jain and Routhier's observations from nearly three decades ago. However, we observe several differences. First, there are new causes of these bursts, including in-NIC mechanisms designed to support higher link rates, such as TCP segmentation offloading. Second, in an effort to reduce CPU load, a number of data center applications rely on batching at various levels, which results in corresponding bursts of packets. Finally, how an application invokes standard OS system calls and APIs has a significant impact on the burstiness of the resulting traffic. Taken together, our aim is to better understand the network dynamics of modern data center servers and applications, and through that information, to enable more efficient packet processing across the network stack.

2. RELATED WORK

Nearly thirty years ago, Jain and Routhier [12] analyzed the behavior of shared bus Ethernet traffic at MIT, and found that such traffic did not exhibit a Poisson distribution. Instead, they found

CoNEXT'13, December 9–12, 2013, Santa Barbara, California, USA.
Copyright 2013 ACM 978-1-4503-2101-3/13/12 $15.00
http://dx.doi.org/10.1145/2535372.2535407.

Batching source	Examples	Reason for batching
Application	Streaming Services (Youtube, Netflix)	Large send sizes
Transport	TCP	ACK compression, window scaling
OS	GSO/GRO	Efficient CPU & memory bus utilization
NIC	TSO/LRO	Reduce per packet head
Disk drive	Splice syscall, disk read-ahead	Maximize disk throughput

Table 1: Sources of network bursts.

that packets followed a "train" model. In the train model, packets exhibit strong temporal and spacial locality, and hence many packets from a source are sent to the same destination back-to-back. In this model packets arriving within a parameterized inter-arrival time are called "cars". If the inter-arrival time of packets/cars is higher than a threshold, then the packets are said to be a part of different trains. The inter-train time denotes the frequency at which applications initiate new transfers over the network, whereas the inter-car time reflects the delay added by the generating process and operating system, as well as CPU contention and NIC overheads. In this paper, we re-evaluate modern data center networks to look for the existence and cause of bursts/trains (two terms that we will use interchangeably).

A variety of studies investigate the presence of burstiness in deployed networks. In the data center, Benson et al. [6] examine the behavior of a variety of real-world workloads, noting that traffic is often bursty at short time scales, and exhibits an ON/OFF behavior. At the transport layer, the widely used TCP protocol exhibits, and is the source of, a considerable amount of traffic bursts [2,4,7,13]. Aggarwal et al. [2] and Jiang et al. [13] identify aspects of the mechanics of TCP, including slow start, loss recovery, ACK compression, ACK reordering, unused congestion window space, and bursty API calls, as the sources of these overall traffic bursts. Previous studies in the wide area include Jiang et al. [14], who examined traffic bursts in the Internet and found TCP's self-clocking nature and in-path queuing to result in ON/OFF traffic patterns. In this paper, we identify the causes of bursty traffic across the stack in a data center setting.

3. SOURCES OF TRAFFIC BURSTS

We now highlight the different layers of the network stack that account for bursts, including the application, the transport protocol, the operating system, and the underlying hardware (summarized in Table 1). Then, in Section 4, we present a detailed analysis of the resulting burst behavior of data center applications.

3.1 Application

Ultimately, applications are responsible for introducing data into the network. Depending on the semantics of each individual application, data may be introduced in large chunks or in smaller increments. For real-time workloads like the streaming services YouTube and Netflix, server processes might write data into socket buffers in fine-grained, ON/OFF patterns. For example, video streaming servers often write 64 KB to 2 MB of data per connection [11,21].

On the other hand, many "Big Data" data center applications are limited by the amount of data they can transfer per unit time. Distributed file systems [5,10] and bulk MapReduce workloads [8,22] maximize disk throughput by reading and writing files in large blocks. Because of the large reads these systems require, data is sent to the network in large chunks, potentially resulting in bursty traffic due to transport mechanisms, as described in the next subsection. NFS performs similar optimizations by sending data to the client in large chunks, in transfer sizes specified by the client.

3.2 Transport

The TCP transport protocol has long been identified as a source of traffic bursts, as described in Section 2. Beyond the basic protocol, some improvements have an impact on its resulting burst behavior. One such example is TCP window scaling (introduced, e.g., in Linux 2.6), in which the TCP window representation is extended from 16-bits to 32-bits. The resulting increase in the potential window allows unacknowledged data to grow as large as 1 GB, resulting in potentially large traffic bursts.

3.3 Operating system

The operating system plays a large role in determining the severity of traffic bursts though a number of mechanisms. First, the API call boundary between the application and OS can result in bursty traffic. Second, a variety of OS-level parameters affect the transport and NIC hardware behavior, including maximum and minimum window sizes, window scaling parameters, and interrupt coalescing settings. Finally, the in-kernel mechanisms Generalized Receive Offload (GRO) and Generalized Segmentation Offload (GSO) can be used to reduce CPU utilization, at the expense of potentially increased burstiness. GSO sends large "virtual" packets down the network stack and divides them into MTU-sized packets just before handing the data to the NIC. GRO coalesces a number of smaller packets into a single, larger packet in the OS. Instead of a large number of smaller packets being passed through the network stack (TCP/IP), a single large packet is passed.

3.4 Hardware

The behavior of hardware on either the sender or receiver affects the burstiness of network traffic. We now review a few of these sources.

Disk drives: To maximize disk throughput, the OS and disk controller implement a variety of optimizations to amortize read operations over relatively slow disk seeks, such as read-ahead caches and request reordering (e.g., via the elevator algorithm). Of particular interest is the interaction between the disk and the *splice()* or *sendfile()* system calls. These calls improve overall performance by offloading data transfer between storage devices and the network to the OS, without incurring data copies to userspace. The OS divides these system call requests into individual data transfers. In Linux, these data transfers are the size of the disk read-ahead, which are transferred to the socket in batches and result in a train of packets. The read-ahead parameter is a configurable OS parameter with a default value of 128 KB.

Segmentation offload: TCP Segmentation Offload (TSO) is a feature of the NIC that enables the OS to send large virtual packets to the NIC, which then produces numerous MTU-sized packets. Typical TSO packet sizes are 64 KB. The benefit of this approach is that large data transfers reduces per-packet overhead (e.g., interrupt processing), thereby reducing CPU overhead.

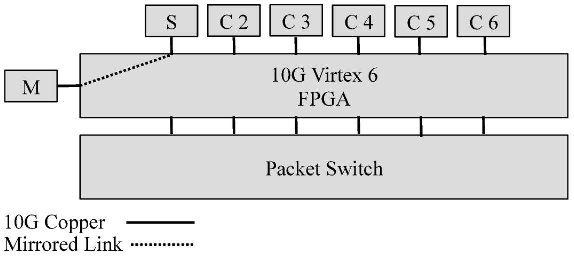


Figure 1: The data center testbed includes an Ethernet packet switch and an FPGA that timestamps packets at a 6.4-ns granularity.

Interrupt Coalescing (IC) and Large Receive Offload (LRO): On the receive side of the NIC driver, IC and LRO both further reduce the CPU overhead of receiving packets at high speed. IC delays receive interrupts until a number of packets have been received, delivering those packets in batches to the OS. LRO combines multiple consecutive packets into a larger, virtual packet in the NIC (as opposed to GRO which combines the packets in the OS). Both potentially affect the burstiness of TCP, as studied in [3, 20, 27].

4. TRAFFIC MEASUREMENTS

To better understand the burst behavior of traffic emanating from hosts in data center environments, we have performed a traffic analysis of a set of testbed servers. Our evaluation seeks to:

1. **Measure the burstiness of a set of data center workloads.** For simple, long-lived communication patterns between a small number of nodes (e.g., a stride pattern), bursts can be quite large—up to about 100 packets. Even for workloads with low destination stability, such as all-to-all patterns, bursts the length of a TSO segment (i.e., 64 KB) are seen in practice.

2. **Understand how each layer of the network stack contributes to burstiness.** We find that application behavior largely determines burstiness, yet even among bandwidth-constrained applications, configuration parameters strongly affect burstiness (for example, NFS configuration parameters or syscall parameters).

3. **Determine the effect of NIC and OS performance features (like GRO and TSO) on traffic emanating from the host.** For bandwidth-constrained workloads, we find that performance enhancing features strongly affect burstiness, especially TSO and LRO.

4. **Determine whether burstiness can be controlled through software changes.** Through modification of the TSO code in the kernel, it is possible to generate larger bursts.

We first describe our measurement methodology and then present our results.

4.1 Measurement methodology

In absence of packet-level traces and measurements on real data centers, we deployed a set of applications on a small, seven-node cluster. The applications we chose are the network file server (NFS), the Hadoop Distributed FileSystem (HDFS), a Hadoop MapReduce-based Terasort, and a set of microbenchmark applications generating synthetic traffic. Each of the applications we evaluate are bandwidth constrained.[1]

Hardware: We deployed the above applications on a set of HP DL360p servers, each with a pair of Intel E5-2630 six-core CPUs (2.3 GHz) running Debian Linux with kernel version 3.6.6. Each server has 16 GB of memory and eight 146 GB 15K RPM hard drives (with an ext3 partition). Each server has an Intel 82599-based 10 Gbps NIC. The measurement server has a Myricom 10G-PCIE2-8B2-2S+E NIC. We used the Intel ixgbe driver (version 3.12.6) with default parameters and multiqueue disabled. With multiqueue enabled, we found that the traffic pattern was dependent on the specific NIC scheduling policy and hence we omitted multiqueue enabled results from our discussion.

OS configuration: The disk read-ahead buffer was configured to be 128 KB with a CFQ disk scheduler. At the start of our experiment, we drop caches to ensure that read requests go directly to the disk. The network interface was configured with a 1500-byte MTU. Unless otherwise noted, in all our experiments we enable both TSO and LRO on the NIC, though we increased the socket buffer size to be 32 MB to avoid bandwidth throttling due to receive window limitations. Linux 3.6.6's default setting of the TCP parameter `tcp_limit_output_bytes` was limiting the number of in-flight TCP packets, which reduced overall performance. Thus, we increased this parameter from 128 KB to 2 MB.

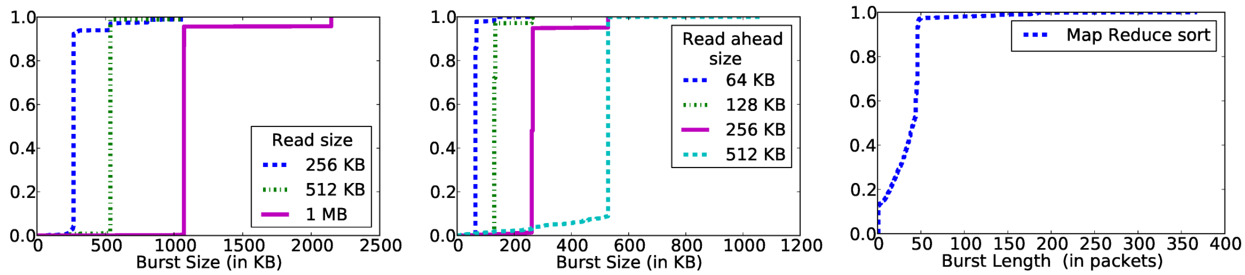
Network and packet capture: Each of the servers is directly connected to a 10 Gbps Xilinx Virtex 6 FPGA. The FPGA "passes through" each connection to ports on a 10 Gbps Fulcrum Monaco packet switch. The purpose of the FPGA is to, on demand, measure the traffic transiting to and from one of the servers. The FPGA generates a measurement record for each packet, including the packet's source and destination address, its size, and a timestamp of when the packet left or arrived to the port. The timing precision of the timestamp is 6.4 ns. The FPGA places per-packet measurements into packets destined to a dedicated measurement server, labeled *M*, as shown in Figure 1. For all the traces we ignore the TCP slow start behavior.

Application configuration: For the NFS experiments, we use NFS server version 3, and the NFS server exports a partition located on a single drive. The in-kernel NFS client reads a file by making multiple pipelined read calls to the server. The read size is specified while mounting the file system and the maximum read size supported by the NFS client is 1 MB. For the HDFS experiments, both the NameNode and the DataNode processes run on the same server, and the DataNode manages data on a single disk per machine. The HDFS data block size is 512 MB. Both the NFS and HDFS servers serve randomly generated files ranging in size from 25 MB to 5 GB. Each client, labeled *C1* through *C6* in Figure 1, copies data from the server to its local disk. We created identical copies of the same file on the server to allow parallel downloads from the clients. To ensure that all the clients start their downloads at the same time, we generate a coordination broadcast packet from a control host.

4.2 Microbenchmarks

To begin our evaluation, we measure a set of synthetic microbenchmarks, which are generated via simple memory-to-memory data transfers to eliminate any effects from the disks, application logic, and think time. The traffic patterns we consider are a *stride* pattern, in which a source host sends data to an intermediate host, which is in turn sending data to a destination host. This pat-

[1] We also evaluated latency-sensitive applications like Memcached, but omit those results since they produced little to no traffic bursts.



(a) NFS server

(b) HDFS DataNode server

(c) MapReduce Sort

Figure 3: An evaluation of three bandwidth-constrained workloads. The Y-axis is CDF. The read-ahead and read syscall sizes largely determine the burst size. For MapReduce workloads, intermediate data shuffling and various keep-alive messages reduce the burst length.

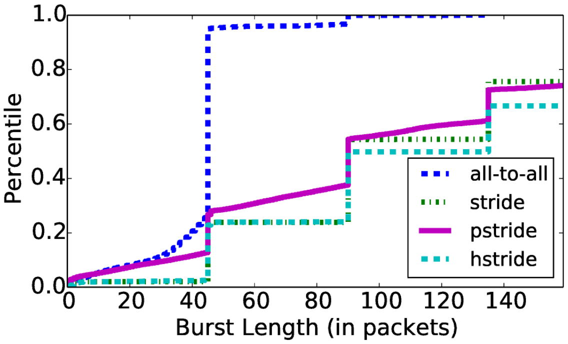


Figure 2: CDF of burst sizes with synthetic traffic patterns (with TSO and LRO enabled; c.f. Figure 4).

tern differs from a simple transfer in that ACKs are intermixed with data packets. Next we consider two variants of the stride workload, both based on workloads used by Farrington et al. [9]. The *pstride* workload is similar to the stride workload, except that each host changes its destination in unison. In the *hstride* workload, each host opens many flows to each destination, and slowly changes the destination host at a flow level. Finally we consider an *all-to-all* workload where every host sends data to every other host.

Figure 2 shows burst lengths in packets. A *burst* is an uninterrupted sequential stream of packets from one source to one destination with an inter-packet spacing of less than 40 ns. The burst length is the number of such closely spaced packets in the stream. We observe that the burst lengths for both the stride and hstride patterns are quite large—up to about 100 packets in the median case. Each of the jumps in the CDF fall at multiples of the TSO size (which in our testbed is 64 KB). This pattern corresponds to the NIC sending one or more TSO-sized amounts of data before switching destination flows. Most surprisingly, in the all-to-all pattern, the median burst length is 44 packets, corresponding to a TSO-sized amount of data. Even though there is no correlation at the application layer, the buffering done via TSO results in bursts of several dozen packets.

4.3 Effect of Application Behavior

We now analyze the burst behavior of three common, bandwidth-intensive applications: NFS, the Hadoop Distributed FileSystem (HDFS), and a MapReduce-based sorting program.

NFS: We configure six NFS clients to concurrently request a file from a single NFS server. Each client specifies the read size at volume mount time, and issues pipelined read requests to the server with a file offset and the read size. Figure 3(a) shows the resulting burst lengths. The length of the packet burst is highly correlated with the underlying read request size configured at the client. This correlation is due to a combination of mechanisms. The NFS server relies on the in-kernel `splice()` system call, which copies data from the disk in configurable batches (e.g., 128 KB by default). The default value of 128 KB can be tuned by changing the disk read ahead size via sysctl. Furthermore, the NFS server also performs its own batching as well, based on the read size that the client specified. As a result, the server reads the entire read size (e.g., 1 MB) off the disk into the memory buffers and then sends the buffered data to the networking stack. Our results confirm that application-layer batching can lead to highly bursty behavior.

HDFS: We next configured six HDFS clients to concurrently request a file from a single HDFS DataNode server. Figure 3(b) shows the resulting burst lengths. As in the NFS application, HDFS-based packet burst sizes are highly correlated with the read-ahead size, since HDFS indirectly uses the `sendfile()` call (via Java's `transferTo()` method). The `sendfile()` call copies data from the disk into the network buffers in batches of disk read-ahead size. Unlike the NFS server, no additional buffering or batching is done by the server process, and so the observed burst lengths correspond more directly to the implementation of `sendfile()`.

MapReduce Sort: To evaluate an all-to-all application we use Hadoop's Terasort. We installed both the NameNode and TaskTracker on node C6, and generated 120 GB of data spread across six remaining nodes (S and C1–C5). The resulting burst behavior is shown in Figure 3(c). The median burst length is approximately 64 packets, which is lower than the file transfer workloads above. Although still large, this lower burst length is due in part to a mixture of flows on each server, since each node exchanges intermediate data, status updates, keep-alive messages, and file transfer requests.

4.4 Effect of NIC Hardware

We now examine several recent mechanisms designed to improve network performance and lower CPU utilization. Specifically, we examine LRO and TSO, deployed in the NIC, and GRO, which runs in the kernel. We repeated the microbenchmark experiments from Section 4.2, enabling or disabling these NIC parameters. Figure 4 shows the results of the all-to-all workload.

We find that enabling TSO affects the burst length directly, whereas LRO, by coalescing several packets on the receive side into larger acknowledged segments, indirectly causes burstiness. Thus, disabling TSO has a more prominent effect on shrinking the burst size, as compared to disabling LRO. However, both strongly affect CPU utilization, as shown in Table 2. Disabling both LRO and TSO drops throughput to 2.4 Gbps, since the kernel cannot keep up with

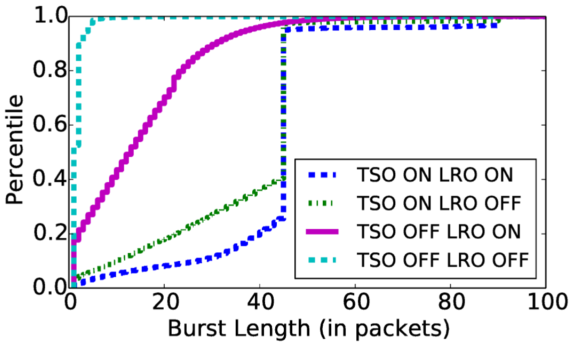


Figure 4: The TSO NIC mechanism directly increases the burst length, while LRO acts indirectly with a smaller effect.

TSO	LRO	Average Throughput
ON	ON	8.7 Gbps
ON	OFF	7.1 Gbps
OFF	ON	5.5 Gbps
OFF	OFF	2.4 Gbps

Table 2: Average throughput with different NIC settings.

the rate of packets necessary to saturate the link. Combinations of LRO and TSO result in intermediate throughputs, and both enabled produce the highest throughput, as expected.

To understand the effect of LRO on burst size, we collected measurements of an *iperf* session between two servers with LRO either on or off, and calculated the throughput, CPU utilization, and ACK ratio for each case. The ACK ratio is the average number of data bytes acknowledged by a single ACK, one indication of burstiness. Table 3 shows these results. Without LRO an ACK is generated for every other segment, whereas with LRO enabled up to 44 KB are acknowledged at a time, resulting in increased burstiness.

Noting that link-level burst lengths are highly correlated with the TSO sizes configured on our NICs, we next seek to understand whether the OS and applications would be able to send larger bursts if the TSO size were increased. Linux is limited to TSO sizes of 64 KB, and the NIC we used supports up to 256 KB. We were able to modify Linux to support a TSO size of up to 148 KB. Figure 5 shows the resulting burst lengths of an one-to-all workload, and indeed the OS and applications were able to send bursts up to this new maximum. Thus traffic leaving a server could be even more bursty by modifying the TSO mechanism.

5. IMPLICATIONS

Thus far, we have shown that traffic exhibits large bursts at sub-100 microsecond timescales, and these bursts are highly correlated with the size of TSO segments, disk read-ahead settings, and application send sizes. Our results indicate that irrespective of higher layer application behavior, packets come out of a 10-Gbps server in bursts due to batching. We now discuss the implications—both positive and negative—of such bursts.

Negative effects of bursts: Blanton and Allman [7] analyze packet traces from three different networks and find that, for large burst lengths (e.g., those greater than 10 segments), the probability of dropping packets increases. In fact, with very large bursts (i.e., 60 segments or more), the probability of dropping a single segment increases to 100%. This connection between packet bursts and packet loss is not confined to these examples. For example, in the YouTube network, bursty traffic is responsible for 40% of total packet losses [21]. Jiang et al. [13] show that flow-level bursts can result in increased queuing delay in the network. Alizadeh et al. [3] show that bursty traffic causes temporary increases in network buffer occupancy, which results in variable latency and higher packet losses within a data center. To reduce burstiness, Blanton and Allman have proposed placing a limit on the sending of new segments in response to an ACK, called *MaxBurst* [4]. Several efforts [2, 3, 28] argue for pacing packets to reduce TCP burstiness.

LRO Setting	CPU Util (%)	Throughput (Gbps)	ACK Ratio (KB)
ON	53	9.49	44
OFF	100	7.3	3
OFF (pinned)	95	9.25	3

Table 3: The effect of LRO on CPU utilization and throughput. The ACK ratio represents the amount of data acknowledged by a single ACK packet, and the “pinned” case refers to a configuration in which the application and interrupts are pinned to separate cores.

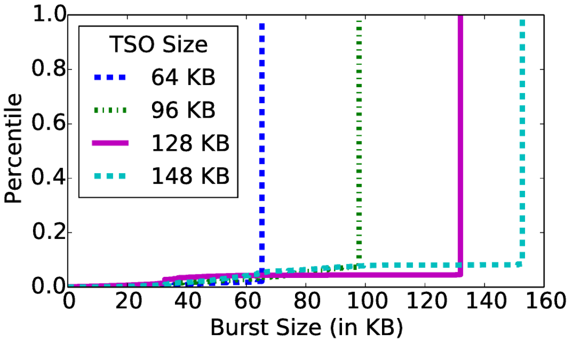


Figure 5: Increasing the TSO size beyond the default maximum of 64 KB results in larger bursts.

Positive effects of bursts: Designing efficient and performant packet processing devices (e.g., middleboxes, switches, SDN controller) relies on having an understanding of the traffic that will transit them. Each of these devices has to carefully manage internal resources including TCAM entries, flow entry caches, and CPU resources, which are affected by the arrival pattern of packets. The presence of bursts show that packets to a destination have temporal locality. Given a packet to a destination, we can predict with some probability that the next packet will be for the same destination. This temporal effect has interesting implications for data center traffic rule and policy management and middle-boxes. Routhier [12] argue that the presence of trains enable certain optimizations in the network, such as amortizing classification operations across a large number of consecutive packets to enable *path caching*. FastTrak [16] and vCRIB [15] exploit locality in flows to offload rules from the hypervisor to the limited memory on hardware NICs and switches. Similarly, software middleboxes (e.g., CoMb [23]) can offload packet classification and policy rules on to the NIC. A CoMb middlebox may also run different applications on the same hardware platform. A CoMb server might exploit burstiness by carefully scheduling applications for better CPU and cache utilization, for example, by running an individual application on an entire burst of packets, followed by the next application, instead of context-switching applications on per-packet basis. Jain and More, and recently, Sinha et al. [24] exploit the burstiness of TCP to schedule "flowlets" on multiple paths through the network. Wischik finds that traffic bursts can potentially inform the sizing of buffers in routers and gateways along the path [26]. Recently, Vattikonda et al. [25] proposed overlaying a data center network with a TDMA-based link arbitration scheme, which replaces Ethernet's shared medium model. With TDMA support, each host would periodically have exclusive access to the link. To maximally utilize that resource, each host would ideally send a burst of data. Porter et al. [19] rely on a similar TDMA scheme to implement an inter-ToR optical circuit switched interconnect.

6. ACKNOWLEDGMENTS

We would like to thank our shepherd, Thomas Karagiannis, and the anonymous reviewers of this work for their valuable feedback. This work was supported in part by grants from the National Science Foundation (CNS-1314921 and CNS-1018808) and by generous research, operational and/or in-kind support from Microsoft, Google, and the UCSD Center for Networked Systems (CNS).

7. REFERENCES

[1] 100Gb/s Ethernet Task Force. `http://www.ieee802.org/3/ba/`.

[2] A. Aggarwal, S. Savage, and T. E. Anderson. Understanding the Performance of TCP Pacing. In *Proc. INFOCOM*, 2000.

[3] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center. In *Proc. NSDI*, 2012.

[4] M. Allman and E. Blanton. Notes on Burst Mitigation for Transport Protocols. *SIGCOMM Comput. Commun. Rev.*, 35(2):53–60, Apr. 2005.

[5] Apache Software Foundation. HDFS Architecture Guide. `http://hadoop.apache.org/docs/hdfs/current/hdfs_design.html`.

[6] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. IMC*, 2010.

[7] E. Blanton and M. Allman. On the Impact of Bursting on TCP Performance. In *Proc. PAM*, 2005.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. OSDI*, 2004.

[9] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *Proc. ACM SIGCOMM*, Aug. 2010.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proc. SOSP*, 2003.

[11] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate Limiting YouTube Video Streaming. In *Proc. ATC*, 2012.

[12] R. Jain and S. Routhier. Packet Trains–Measurements and a New Model for Computer Network Traffic. *IEEE J.Sel. A. Commun.*, 4(6):986–995, Sept. 1986.

[13] H. Jiang and C. Dovrolis. Source-level IP Packet Bursts: Causes and Effects. In *Proc. IMC*, 2003.

[14] H. Jiang and C. Dovrolis. Why is the Internet Traffic Bursty in Short Time Scales? In *Proc. SIGMETRICS*, 2005.

[15] M. Moshref, M. Yu, A. Sharma, and R. Govindan. Scalable rule management for data centers. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 157–170, Berkeley, CA, USA, 2013. USENIX Association.

[16] R. N. Mysore, G. Porter, Subramanya, and A. Vahdat. FasTrak: Enabling Express Lanes in Multi-Tenant Data Centers. In *Proc. CoNEXT*, 2013.

[17] ONF. Software-Defined Networking: The New Norm for Networks. `https://www.opennetworking.org/`.

[18] Open vSwitch: An Open Virtual Switch. `http://openvswitch.org/`.

[19] G. Porter, R. Strong, N. Farrington, A. Forencich, P.-C. Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat. Integrating Microsecond Circuit Switching into the Data Center. In *Proc. SIGCOMM*, 2013.

[20] R. Prasad, M. Jain, and C. Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *Proc. PAM*, 2004.

[21] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network Characteristics of Video Streaming Traffic. In *Proc. CoNEXT*, 2011.

[22] A. Rasmussen, G. Porter, M. Conley, H. V. Madhyastha, R. N. Mysore, A. Pucher, and A. Vahdat. TritonSort: A Balanced Large-Scale Sorting System. In *Proc. NSDI*, 2011.

[23] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, , and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proc. NSDI*, 2012.

[24] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCPs Burstiness using Flowlet Switching. In *Proc. HotNets*, 2004.

[25] B. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical TDMA for Datacenter Ethernet. In *Proc. ACM EuroSys*, 2012.

[26] D. Wischik. Buffer Sizing Theory for Bursty TCP Flows. In *Communications, 2006 International Zurich Seminar on*, pages 98–101, 2006.

[27] T. Yoshino, Y. Sugawara, K. Inagami, J. Tamatsukuri, M. Inaba, and K. Hiraki. Performance Optimization of TCP/IP over 10 Gigabit Ethernet by Precise Instrumentation. In *Proc. SC*, 2008.

[28] L. Zhang, S. Shenker, and D. D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proc. SIGCOMM*, 1991.