Sharing Bandwidth by Allocating Switch Buffer in Data Center Networks

Jiao Zhang, Fengyuan Ren, Xin Yue, Ran Shu, and Chuang Lin

***Abstract*—In today's data centers, the round trip propagation delay is quite small. Therefore, switch buffer sizes are much larger than the Bandwidth Delay Product (BDP). Based on this observation, in this paper we introduce a new transport protocol which provides bandwidth Sharing by Allocating switch Buffer (SAB) for data centers. SAB sets the congestion windows for flows based on the buffer size of the switches along the path. On one hand, as long as the total buffer allocated to all the flows is larger than the BDP, the network bandwidth can be fully utilized. On the other hand, since SAB only allocates the buffer space to flows, the totally injected traffic will not exceed the network capacity. Thus, SAB rarely loses packets. SAB also reduces flow completion time by allowing flows to reach their fair share of bandwidth quickly. The results of a series of experiments and simulations demonstrate that SAB has the features of fast convergence and rare packet loss. It reduces the latency of short flows and solves theTCP Incast and TCP Outcast problems.**

***Index Terms*—Data Center Networks, Transport Protocol, Fast Convergence, Rare Loss**

I. Introduction

DATA center networks have extremely low Round Trip Time (RTT), typically on the order of a few hundred microseconds [1], [2]. This low RTT results in a small bandwidth-delay product. Therefore, the switch buffer size is often larger than the in-flight traffic on the links. Also, applications in data centers incur a large number of short messages [3], [4], high churn [5], and special communication patterns, such as many-to-one [6], [7].

Due to these features of data centers, traditional transport protocols like TCP face many challenges. The three main challenges are: high latency of short flows [3], [8], TCP Incast [1], [7], [9] and TCP Outcast [10]. The TCP Incast problem occurs when multiple senders synchronously transmit stripe units to a single receiver. The synchronized multiple connections can easily cause traffic bursts which cause timeout periods. Since the Minimum Retransmission TimeOut (RTOmin) of TCP generally equals 200 milliseconds in default, which are orders of magnitude larger than the microsecond-granularity RTT in data center networks, the bandwidth waste during timeout periods dramatically decreases goodput. The TCP Outcast [10] refers to the problem that when two groups of flows arrive at two input ports, and need to be forwarded to the same output port, the smaller group, on average, has lower throughput.

Several solutions have been proposed to deal with these problems. Researchers at Google suggested increasing TCP's initial window to at least 10 packets [8] or enabling data transmission during TCP's initial handshake phase [3] to decrease the response time of web queries. M. Alizadeh *et al.* developed DCTCP [4] which leverages the ECN mechanism to keep the queue small and thus decrease the delay of short flows. To mitigate the TCP Incast throughput collapse, V. Vasudevan *et al.* suggested decreasing RTOmin to reduce the bandwidth waste caused by timeouts [1] and W. Hu *et al.* proposed ICTCP, which adjusts the receiver window to avoid packet loss [11]. As for the TCP Outcast problem, it is shown that some queue management schemes (such as RED [12] and SFQ [13]) and assisted or enhanced mechanisms (such as TCP pacing and equal-length routing) can improve fairness to some extent, but hardly eradicate the problem [10].

Existing solutions address these problems individually, which means that while the solutions benefit some specific applications, they might negatively impact others. For example, increasing TCP's initial window size exacerbates the throughput collapse in the Incast scenarios due to the heavy bursts introduced by the larger initial window, and reducing RTOmin readily leads to spurious timeout.

What are the desirable properties for a data center transport protocol? First, *fast convergence*. The convergence rate of traditional TCP protocols is so slow that the short flows, which only consist of several packets, are transmitted during the climb-up phase of the congestion window in the slow start procedure. The sluggishness of the flow control algorithm defers the transmission of short messages. Second, *losses should be rare*. In some specific applications, the loss of some packets incurs high cost. For example, if any one of the last three packets of a block is lost in a TCP Incast scenario, timeout will be unavoidable due to inadequate ACKs [7], [9]. The TCP Outcast problem is also caused by packet losses. Thus, the flow control algorithms driven by packet dropping are unsuitable for data center networks. Ideally, a transport protocol without packet loss is desired. However, it is very hard to guarantee zero packet loss ratio without quite complex control. For example, the credit-based flow control avoids packet loss at the cost of maintaining buffer states for every Virtual Channel (VC) at all the routers/switches [14]. For a VC, only if the corresponding buffer has available space, the sender could get some credits and transmit the same number of packets. In data centers, the number of connections is quite large, it is impractical to employ the zero-loss transport protocol without involving too much complexities. However, the

Manuscript received January 15, 2013; revised July 1, 2013.

J. Zhang, F. Ren, R. Shu and C. Lin are with Dep. of Computer Science and Technology, Tsinghua Univ., Beijing, China and Tsinghua National Laboratory for Information Science and Technology, China (e-mail: zhangjiao1986@gmail.com, {renfy, shuran, clin}@csnet1.cs.tsinghua.edu.cn).

X. Yue is with Dep. of Electronic Engineering, Tsinghua Univ, (email: yuecn41@gmail.com).

Digital Object Identifier 10.1109/JSAC.2014.140105.

loss ratio should be reduced to the greatest extent. Therefore, a transport protocol with small overhead and rare loss is desired.

Can one design a new congestion control mechanism that runs on the end hosts and possesses both properties of fast convergence and rare packet loss? Small RTTs, high churn and many-to-one communication pattern mean that the congestion windows of short flows almost never achieve the appropriate values. Therefore, it is hard to design a congestion control mechanism at end hosts to satisfy all the requirements of the applications in data centers.

In this work, by taking advantage of the specific feature of data center networks, namely, quite small round trip propagation delay, we propose a transport protocol which *allocates the switch buffer* to determine the congestion window of each flow.

Ideally, the bottleneck link should be full of packets and the switch buffers should have quite small queue length, that is, the total amount of the injected traffic approximately equals the in-flight capacity. The switch buffer is reserved to accommodate traffic bursts. Driven by packet losses, TCP and most of other transport protocols probe the maximum network capacity, including the buffer space and the in-flight capacity, which naturally results in lots of packets dropping.

Based on the observation that the switch buffer size is much larger than the BDP in data center networks, we follow the framework of sliding window flow control defined in the standard TCP, and customize a brand new flow control mechanism for data center networks. The proposed mechanism dynamically and promptly adjusts the congestion windows of flows by allocating the switch buffer associated with the bottleneck links. We referred to it as SAB.

SAB has two main advantages. First, it converges fast. Because switches explicitly determine the congestion window for each flow, SAB flows can converge to their fair bandwidth in one RTT. Therefore, the latency of short flows can be greatly reduced. Note that since SAB is a window-based congestion control mechanism instead of rate-based, the fair throughput achieved by a flow is inversely proportional to the RTT of the flow. This kind of fairness with RTT bias also happens in TCP according to the throughput model of a TCP flow [15]. Second, SAB rarely loses packets. The maximum number of packets accommodated by a network during one RTT is the summation of the buffer size and the in-flight capacity. SAB only allocates the buffer space to the senders. Besides, SAB adapts to the situation where no more than one packet is allocated to each flow by reducing MSS (Maximum Segment Size). Hence SAB rarely loses packets. This feature can solve the goodput collapse of TCP Incast as well as the unfairness of TCP Outcast since they are both caused by large numbers of packet losses.

We implemented the end host part of SAB in Linux kernel 2.6.38.3 and the switch part in NetFPGA. The results of our experiments show that SAB indeed converges fast and rarely loses packets. Furthermore, to evaluate the scalability of SAB, we conducted a series of simulations on the ns-2 platform with higher bandwidth and more senders.

The remainder of the paper is organized as follows. In the next section, the features of related work are summarized. The design rationale of SAB is presented in Section III. Section IV describes the SAB mechanism in detail. In Section V, the performance of SAB is evaluated in several designed scenarios and an emulated real scenario. The results are compared with DCTCP, D^{2}TCP and TCP NewReno (with SACK). In Section VI, we show the simulation results on the ns-2 platform with higher bandwidth and more senders. The paper is concluded in Section VII.

II. Related Work

Credit-based Flow Control. Kung *et al.* proposed a credit-based flow control mechanism without packet loss for ATM networks [14], [16]. Each Virtual Channel (VC) is assigned a constant or dynamic buffer space. The receiver sends credits to the sender if some buffer space is available. Therefore, the credit-based flow control mechanism prevents packet loss. If the number of connections that share a path is small, this method will not introduce much overhead. However, communications among thousands of servers are quite frequent in data centers, the number of connections along a path is usually quite large. Maintaining the buffer space for each connection at all the switches will introduce high overhead.

Evolution-based Flow Control. The number of literature on TCP and its variants is huge. Here we mainly summarize the relevant protocols proposed for data center networks.

To achieve high throughput for long flows as well as small delay for short ones, DCTCP [4] employs the Explicit Congestion Notification (ECN) and modifies the TCP congestion control mechanism at senders to maintain a small switch queue length. D^{2}TCP [17] extends the window evolution algorithm of DCTCP to provide differentiated services. The flows with smaller remaining delay can get higher congestion window size. The remaining flow size and the remaining time of deadline-aware flows are required in this protocol. ICTCP is proposed to solve the TCP Incast problem by adjusting the advertised window (*awnd*) at receivers [11]. The bottleneck link is assumed to directly connect to the receiver. The receiver estimates the available bandwidth and RTT to compute the reasonable *awnd* and thus each flow fairly injects proper traffic to the network. However, exactly estimating the available bandwidth and RTT in real-time is difficult. Besides, ICTCP fails to work well if congestion does not happen at the last hop.

Seawall aims to solve the performance interference and denial of service attacks among tenants in virtual data centers [5], [18]. A. Shieh *et al.* proposed an edge-based solution which achieves max-min fairness across tenant VMs by sending traffic through hypervisor-to-hypervisor tunnels. Each tenant needs to be assigned a globally appropriate weight, which is a challenging work. TCP Fast Open is designed to improve the response rate of web services [3]. Data can be exchanged during TCP's initial handshake phase. Similar idea has been proposed in earlier work to reduce latency [19], [20]. TCP Fast Open deceases HTTP transaction network latency by 15%. In [8], The initial window of TCP is increased to 10 MSS to reduce the number of RTTs required to finish a web message. However, it will exacerbate the throughput collapse of TCP Incast due to quite large initial window. Besides, a constant

higher initial congestion window could not adapt to different network environments.

Explicit Rate Allocation. XCP [21] is a typical congestion control scheme of explicit rate allocation. Each packet header is extended to include current RTT and cwnd. Routers use the information of all the flows to allocate bandwidth. XCP can achieve high link utilization. However, it typically results that the flows last two orders of magnitude longer than necessary when there is a mix of flow sizes [22]. RCP [22] solves the problem of XCP. However, similar to XCP, each packet needs to carry the RTT value in its header. The router needs the RTT information to compute that how much bandwidth is required to drain the current accumulated packets in the queue during a RTT period. The time-varying and quite small RTT is hard to be exactly estimated in data centers. Most kernel implementations track RTTs at a granularity of 1 ms (the period of a jiffy), which is larger than the RTT in data center networks.

In data center networks, D^3 is designed to satisfy the delay requirement of some flows [23]. When initiating a deadline-aware flow, the source requests a desired rate based on the flow size and deadline. Switches determine an allocated rate for each flow. To adjust to the network state and satisfy the deadline requirements, sources need to send update messages to switches to announce their new desired rates periodically, which will bring some overhead. Besides, D^3 operates under the assumption that the deadline is hard. However, in practice, possibly it is better that all the synchronized flows complete at the same time rather than several of them finish earlier in some applications, such as the services with barrier synchronized traffic pattern [11], and some MapReduce-based applications.

III. Design Rationale

Our goal is to design a new transport protocol by making use of the special feature of small round trip propagation delay in data centers. Small propagation delay is a well-known and important feature of data centers, and is the reason why TCP timeouts incur dramatic goodput collapse in the Incast communication pattern [7]. Small propagation delay allows only a few packets to be on the link. For example, in a data center network with link capacity $C = 1$ Gbps and round trip propagation delay $D = 200$ us, the in-flight traffic only equals $CD = 25$ KBytes.

Compared with the small in-flight capacity, the switch buffer size in data center networks is relatively large [4]. Therefore, if Van Jacobson's pipe model [24] is followed to design the transport protocol in data centers, most of packets injected by the senders will be cached in the switch buffers.

Enlightened by the special feature of data centers and considering the challenges of designing congestion control mechanisms at end hosts, we intend to customize a transport protocol for data centers, which promptly adjusts congestion window sizes by properly allocating switch buffers. Next we will demonstrate our idea using a simple model.

Consider a scenario where N flows share a bottleneck link with capacity C bytes per second. The bottleneck buffer size is B bytes. The round trip propagation delay of flow i is denoted as D_i. Let W_i represent the congestion window of flow i. The following proposition can be obtained.

PROPOSITION 1: If the summation of the congestion window of all the flows satisfies

$$\sum_{i=1}^{N} W_i \in [CD_{max}, CD_{min} + B] \tag{1}$$

where $D_{max} = \max\{D_i\}$ and $D_{min} = \min\{D_i\}$, the bottleneck link can be fully utilized without packet losses.

Proof: The bottleneck link can be fully utilized as long as there are always packets to be sent in the bottleneck queue.

The bottleneck queue length equals the totally injected packets minus the in-flight packets on the links. Denote N_f as the number of in-flight packets. The queue length $Q = \min\{(\sum_{i=1}^{N} W_i - N_f)^+, B\}$, where function $(a)^+$ equals a if a is a positive number, otherwise equals 0.

Next we firstly prove the proposition in the simple case that all the flows have equal round trip propagation delay, then we consider the case that the flows suffer different latencies.

- The flows have the same round trip propagation delay D.

In this situation, the in-flight capacity $N_f = CD$. Obviously, if $\sum_{i=1}^{N} W_i < CD$, then the link will be idle at some time. Thus, the link can not be fully utilized. While when $\sum_{i=1}^{N} W_i > CD + B$, some packets will be lost at the bottleneck buffer. If $\sum_{i=1}^{N} W_i \in [CD, CD + B]$, the buffer always has packets to be scheduled and does not drop packets since the rate of arrival packets does not exceed the link capacity.

- The flows have different round trip propagation delay D_i.

From the above analysis, we know that as long as $0 \leq \sum_{i=1}^{N} W_i - N_f \leq B$, the bottleneck link can be fully utilized without packet losses. If the flows have different round trip propagation delay, then the number of the in-flight packets $N_f \in [CD_{min}, CD_{max}]$. To satisfy $\sum_{i=1}^{N} W_i - N_f \geq 0$, $\sum_{i=1}^{N} W_i$ should be larger than the maximum of N_f, that is, $\sum_{i=1}^{N} W_i \geq \max\{N_f\} = CD_{max}$. Besides, to satisfy $\sum_{i=1}^{N} W_i - N_f \leq B$, the inequality, $\sum_{i=1}^{N} W_i \leq N_f + B$, should be satisfied. Since the minimum of $(N_f + B)$ is $(CD_{min} + B)$, when $\sum_{i=1}^{N} W_i \leq CD_{min} + B$, $\sum_{i=1}^{N} W_i \leq N_f + B$ can be satisfied.

Thus, we can infer that as long as the summation of the congestion windows of all the senders is between CD_{max} and $(CD_{min} + B)$, the aggregated throughput can achieve the maximum capacity of the bottleneck link. ■

Since the switch buffer size B is much larger than the product of the link capacity and the round trip propagation delay in data center networks, if the buffer space B is assigned to all the senders, obviously the bottleneck link can be fully utilized. Besides, the fact that the allocated buffer size never exceeds $(CD_{min} + B)$ confirms that the injected packets will not overwhelm the buffer associated with the bottleneck link.

Figure 1 shows a simple example to illustrate the basic idea of SAB. Several senders transmit packets to some receivers. Figure 1(a) shows that part of the bottleneck buffer space is fairly divided among the passing flows. The congestion window value can be conveyed by the headers of data packets and then be returned to the senders by the headers of ACKs. Figure

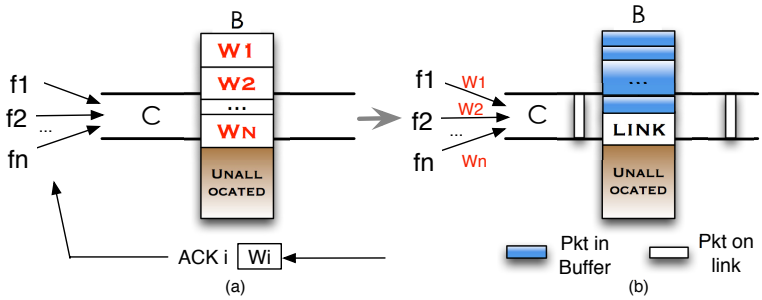


Fig. 1. Basic idea of SAB. (a) Part of buffer space is fairly allocated to all the flows. (b) The flows inject packets according to their congestion windows.

1(b) illustrates that each sender injects packets according to the allocated congestion window size. Due to the fact that the BDP is much smaller than the allocated buffer size in data centers, only a small proportion of the injected packets are on the link, and most of them are accumulated in the switch buffer. Thus, the bottleneck link will not be idle. The allocated buffer space can be adjusted to keep full link utilization as well as low queuing delay.

This fair allocation ensures that each short flow could quickly obtain its fair share of bandwidth. In TCP, the congestion window size of a long flow could achieve a quite large value under the AIMD window evolution algorithm, while a short flow only achieves a quite small congestion window size due to its short duration. Long flows take up a large occupancy of the bottleneck switch buffer, which causes that the packets of short flows have to wait for a long time before being forwarded by the switch. Generally, short flows, such as web query flows, are delay-sensitive, while long flows, such as file backup, can endure a period of latency [4]. Therefore, long flows should give way to the short flows. The shortest job first scheduling mechanism can satisfy this requirement. However, it possibly causes the starvation of long flows.

SAB solves the problem well. It ensures that short and long flows have the same congestion window in only one RTT. Since long flows only take a quite small percentage in data centers, less than one percent [25], the queuing delay caused by long flows is small. Meanwhile, since the long flows are transmitting packets all the time, they do not suffer starvation.

IV. Details of SAB

SAB is a window-based congestion control protocol like TCP. The difference is that in TCP each sender dynamically adjusts its window according to the network congestion feedback, while in SAB the switch assigns congestion window to the passing flows.

Based on the analysis in the above section, if $B > CD_{max}$, we could fairly allocate B to all the senders. However, the simple allocation will lead to large queuing delay since most of the allocated packets will be cached in the queue. Therefore, in SAB, we let $\varepsilon B (0 < \varepsilon \leq 1)$ be the available buffer space for all the senders.

A. Protocol

SAB does not modify the TCP header. The *window* field in the TCP header is designed to convey the advertised window (*awnd*). In SAB, the field is used to convey the minimum of *awnd* and the congestion window assigned by the switches.

1) Sender: Before transmitting a data packet, the sender modifies the *window* field of the packet header to be *0xffff* as the initial window value. After receiving an ACK, according to the window value taken by the ACK header, the sender determines its congestion window.

2) Switch: The main task of the switches is computing the congestion window for each passing flow. To do this, each switch requires to maintain the number of passing flows, N, of each port, and modify the *window* field of passing data packets to be the value calculated by the switch if needed. In SAB, we use the handshake messages, SYN and FIN, to maintain the number of flows N. N increases or decreases by one after receiving a SYN or FIN message, respectively. As mentioned at the beginning of this section, to keep high utilization while small queuing delay, the bottleneck switch allocates $\frac{\varepsilon B}{N}$ buffer space to each sender. Thus, when a data packet passes through a switch, if the value in its *window* field equals *0xffff* or is larger than $\frac{\varepsilon B}{N}$, then the switch needs to update the value of the *window* field to be $\frac{\varepsilon B}{N}$.

3) Receiver: The function of SAB at the receiver side is almost the same as that of TCP. The only difference is that in SAB, before transmitting an ACK for a data packet, the minimum of *awnd* and the *window* value in the header of the data packet is assigned to the *window* field of the ACK header.

B. Congestion Window of Less Than One

One special situation that less than one packet is allocated to each sender when the number of senders is quite large should be considered. Most of the prior work fails to properly deal with this issue, such as DCTCP [4], D^{2}TCP [17], ICTCP [11]. However, this situation indeed exists. Besides, it will become more common in the large-scale data centers.

There are two main solutions to the problem. First, source i sends one packet every $\frac{1}{W_i}$ RTTs. To implement this solution, exactly estimated RTT and a high resolution timer to count $\frac{RTT}{W_i}$ are needed. Precise estimating RTT is difficult since RTT is quite small. Second, transmitting one packet in a RTT randomly selected from $\frac{1}{W_i}$ ones. However, this method makes the self synchronization mechanism fail. For example, if the

congestion window of sender i is 0.5, then during each RTT, no packet will be transmitted with probability of 0.5. If no packet is transmitted during RTT j, then the sender will not receive ACK any more. The self synchronization mechanism fails.

To avoid these drawbacks, the MSS is proportionally reduced according to the congestion window at each sender in the SAB algorithm. If the congestion window becomes larger than 1 later, the MSS will go back to the normal value. This simple method effectively avoids the congestion caused by too many concurrent flows.

C. Choosing Parameter ε

The parameter ε determines the proportion of the assigned buffer space. From eq. (1), we can infer that the assigned buffer space εB should satisfy $\varepsilon B \geq CD_{max}$. Besides, ε is not larger than 1. Therefore, ε should take values as follows:

$$\frac{CD_{max}}{B} \leq \varepsilon \leq 1 \tag{2}$$

Note that eq. (2) assumes that the count of the number of flows is accurate. However, precise counting will incur large overhead if the number of flows is large. There are some literature on how to estimate the number of active flows using little memory with small estimation error [26], [27], and fortunately SAB can endure the estimation error within a certain range by adjusting the parameter ε. Next we will analyze how to set the parameter ε if the number of flows, N, is estimated with some error.

Let $\hat{N}$ be the estimated number of active flows passing a switch. Define the estimation error η as $\eta = \frac{|N-\hat{N}|}{N}$.

PROPOSITION 2: Given that the estimation error of the active flows is η, the parameter ε should satisfy

$$(1+\eta)\frac{CD_{max}}{B} \leq \varepsilon \leq \min\{1, (1-\eta)\frac{B+CD_{min}}{B}\}. \tag{3}$$

The allowed maximum of the estimation error, η_{max}, in SAB is

$$\eta_{max} = \frac{B + CD_{min} - CD_{max}}{B + CD_{min} + CD_{max}}. \tag{4}$$

Proof. See Appendix A.

Given the upper bound of the estimation error of the method that counts the number of active flows, we can set proper ε. The allowed maximum estimation error in SAB is usually quite large. For example, if C=1 Gbps, D_{max}=300 us, D_{min}=100 us, B=512 KB, then CD_{min}=12.5 KB, CD_{max}=37.5 KB. Therefore, we have $\eta_{max} = \frac{512+12.5-37.5}{512+12.5+37.5} = 0.8665$.

Probability of Packet Loss. Since SAB can endure the estimation error as high as η_{max}, it can be inferred that SAB does not lose packets unless the estimation error exceeds η_{max}. Therefore, the probability of packet loss is $Pr\{\text{drop}\} = Pr\{\frac{\hat{N}-N}{N} > \eta_{max}\}$.

D. Work Conserving

In SAB, each flow injects packets according to the congestion window assigned by switches. We have demonstrated that it can achieve throughput as high as the maximum capacity of the bottleneck link in scenarios with one bottleneck. When multiple bottlenecks exist, is SAB work-conserving? that is, whether SAB can fully utilize the available bandwidth at all the bottlenecks?

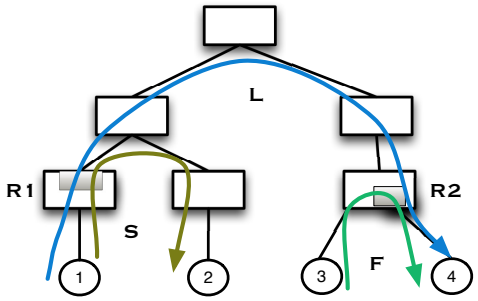


Fig. 2. A scenario with multiple bottlenecks. The grayed ports are bottlenecks. S, L, F represents the sets of flows from $1 \rightarrow 2$, $1 \rightarrow 4$, $3 \rightarrow 4$, respectively.

Consider the scenario in Figure 2. Let N_s, N_l, N_f be the number of flows in flow sets S, L, F, respectively. Assume that R_1 and R_2 have the same buffer size.

Obviously, if $N_s = N_f$, the congestion window sizes of all the flows are the same as that with only one bottleneck. The capacity of links connecting to R_1 and R_2 can be both fully utilized. However, if $N_s \neq N_f$, eg. $N_s < N_f$, R_1 will possibly be under-utilized since the N_l flows inject fewer packets than that allocated by R_1. Next we will show under what conditions both of them can be fully utilized.

Denote $N_l = aN_s = bN_f$ $(a > b)$. If the estimated number of flows is precise and the estimation error tolerance is η. The following proposition can be established.

PROPOSITION 3: If a and b satisfy

$$\frac{1}{1+a} + \frac{b}{1+b} > \frac{1}{1+\eta}, \tag{5}$$

then both of the bottlenecks can be fully utilized.

Proof. See Appendix B.

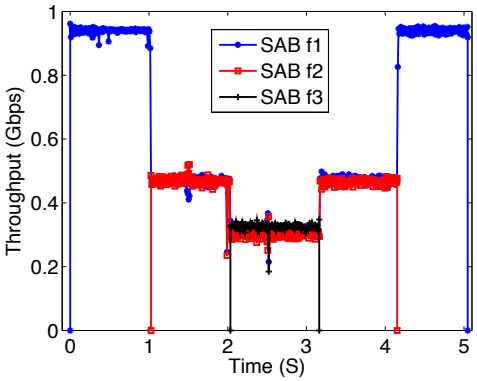
Correspondingly, if $a < \eta$ or $\frac{1}{b} < \eta$, then eq. (5) is satisfied, that is, when the ratio of the number of long flows N_l to that of N_s is smaller than η, or the ratio of N_l to N_f is larger than η, the under-utilization phenomenon will not happen. If both a and $\frac{1}{b}$ are larger than η, then a and b should satisfy that $\frac{a-b}{1+2b+ab} < \eta$.

V. Experimental Evaluation

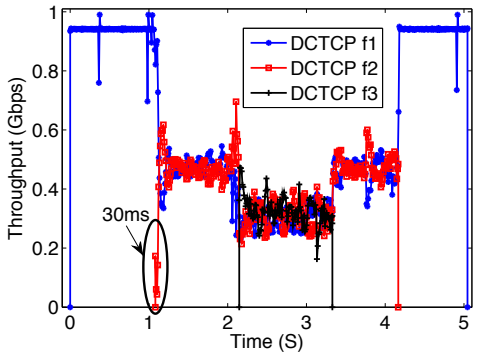
A. Experiment Setup

We set up different topologies to comprehensively test the performance of SAB. The servers used in all the experiments are DELL OptiPlex 360 desktops with Intel 2.93 GHz dual-core CPU, 6 GB DRAM, 300 GB hard disk, and Intel Corporation 82567LM-3 Gigabit Network Interface Card. The operating system is CentOS-5.5. Each NetFPGA card hosting in a DELL server has a buffer of 512 KB per port and four 1 Gbps ports. The output-buffer management scheme is used.

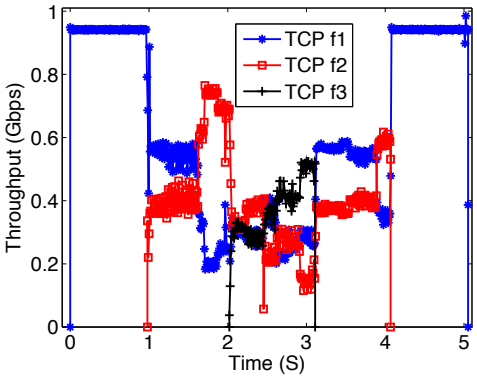
We validate the performance of SAB by comparing it with TCP NewReno (with SACK), DCTCP and D^{2}TCP. The reason of choosing these three protocols is that TCP NewReno is widely used in practice [28], and



(a) SAB



(b) DCTCP



(c) TCP

Fig. 3. Convergence and Fairness.

DCTCP as well as D^{2}TCP are recently proposed well-known transport protocols for data center networks. TCP NewReno (with SACK) is implemented in the kernel 2.6.38.3 by configuring *net.ipv4.tcp_congestion_control=reno* and *net.ipv4.tcp_sack=1*. For DCTCP, we run the code shared on-line [29]. D^{2}TCP is implemented by modifying DCTCP. The parameter d is limited between 0.5 and 2 as stated in D^{2}TCP [17]. For SAB and TCP, the queue management scheme is DropTail. The threshold of marking packets in DCTCP and D^{2}TCP is 32 KB as recommended in [4]. In SAB, ε is set to $\frac{1}{2}$.

Firstly, we evaluate the performance of the proposed SAB in terms of the basic metrics of a transport protocol, including throughput, convergence, fairness and so on. Then we show

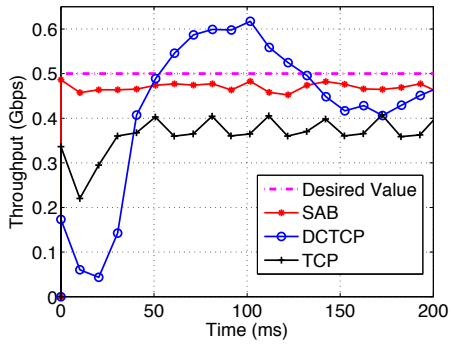


Fig. 4. A long lived flow exists. The throughput variation of the second flow.

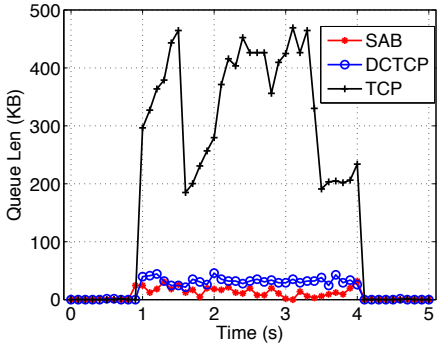


Fig. 5. Queue length of bottleneck buffer in the fairness scenario.

that SAB can reduce the latency of flows as well as solve the TCP Incast and Outcast problems due to its advantages of fast convergence and rare loss. In these scenarios, the flows do not have deadline requirements. Thus, D^{2}TCP performs the same as DCTCP, and its results are omitted. At last, the performance of SAB is compared with TCP, DCTCP and D^{2}TCP in our testbed with the practical traffic, which is generated according to the traffic characteristic in DCTCP [4]. Each query flow and short message has a deadline that is proportional to the flow size. The background flows do not have deadlines.

B. Results

1) Basic metrics: First of all, we evaluate the three basic metrics of a transport protocol, throughput, convergence and fairness. We connect 4 hosts to a NetFPGA switch via 1 Gbps links. One of them acts as the receiver, while the others sequentially connect to the receiver at the interval of one second. The last started flow will be stopped first. The throughput results are drawn in Figure 3. The throughput is sampled every 10 milliseconds. We can see that the throughput values of all the three protocols are close to the maximum of the link capacity. However, in terms of the convergence rate, *SAB flows converge to their fair share of bandwidth more quickly*. The DCTCP and TCP flows, shown in Figure 3(b) and 3(c) respectively, can also achieve their fair throughput, but the throughput curves shake up and down. Especially the throughput values of TCP flows fluctuate largely. DCTCP flows converge more slowly than SAB flows do.

To show the detail more clearly, we magnify the throughput curve of 200 milliseconds from the start of flow 2 in Figure

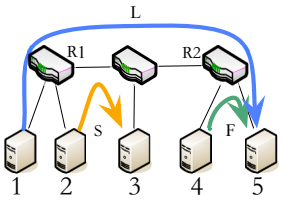


Fig. 6. Multi-Bottlenecks: topo. S, L, F represents flows $2 \rightarrow 3$, $1 \rightarrow 5$, $4 \rightarrow 5$. R_1 and R_2 are bottlenecks.

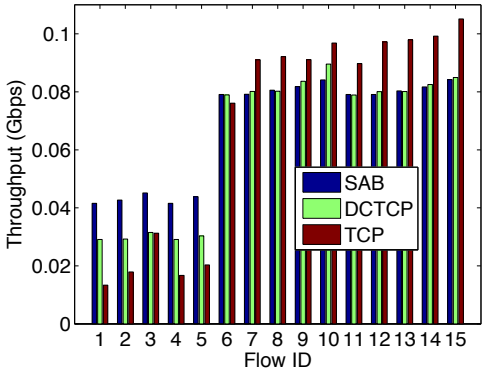


Fig. 7. Multi-Bottlenecks: Throughput of the flows passing bottleneck buffer at switch R1.

4. DCTCP spends 50 milliseconds in evolving to the desired value, i.e., 0.5 Gbps, which is too long for short flows. Since most of them can be finished in 10 milliseconds [4]. Besides, the congestion window size in DCTCP overshoots its fair share since the throughput continually increases for a long time (about 50 ms) instead of staying at the ideal value after arriving at it. This is possibly because of the unfair marking on different flows in DCTCP. The TCP flow converges quite slowly. While the SAB flow immediately converges to the expected throughput.

Figure 5 depicts the queue length evolution. At the first and last 1 second, none of the three queues has backlog. During the 1-4 seconds, coexistence of multiple flows leads to backlog in queues. TCP probes the maximum of the buffer size per port and then decreases to about half of it. While both of DCTCP and SAB have small and stable queue length.

2) Multiple bottlenecks: To evaluate whether SAB performs well in multiple bottlenecks, we use the topology shown in Figure 6 to conduct a series of experiments. Figure 7 and Figure 8 present the throughput results of all the flows at switch R_1 and R_2 when $N_s = 10, N_l = 5, N_f = 15$. In Figure 7, the flows with IDs from 1 to 5 represent 5 long flows, and the remainder 10 flows belong to flow set S. We can see that the throughput difference of two TCP flows is larger than that in SAB and DCTCP.

In Figure 8, flows with IDs from 1 to 5 are the long flows and the flows with IDs from 6 to 20 belong to flow set F. The difference between the throughput of each long flow and each short flow is not large in SAB. This is because the RTT ratio of a long flow and a short flow is $(8\times linkDelay+\frac{Q(R_1)}{C}+\frac{Q_{R_2}}{C}) : (4\times linkDelay+\frac{Q_{R_2}}{C})$, where $\frac{Q(R)}{C}$ denotes the queuing delay of R. Since the queuing delay at switch R_1 is smaller than

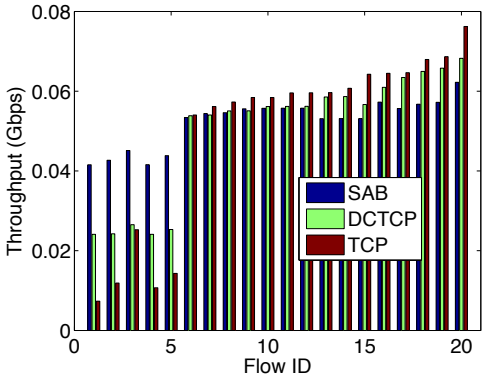


Fig. 8. Multi-Bottlenecks: Throughput of flows passing bottleneck buffer of R2.

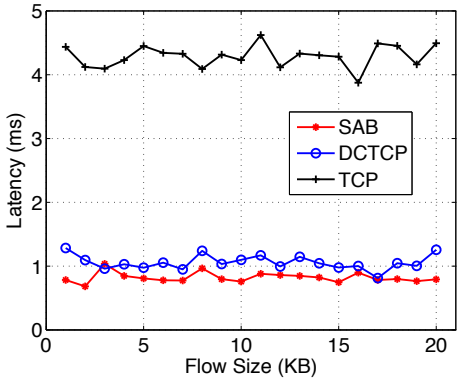


Fig. 9. Delay of different protocols with different flow size and 30 background flows.

TABLE I
AGGREGATED THROUGHPUT AT SWITCHES R_1 AND R_2

	SAB	DCTCP	TCP
R1	0.99 Gbps	0.93 Gbps	0.99 Gbps
R2	0.99 Gbps	0.95 Gbps	0.94 Gbps

that at switch R_2, the RTT ratio of a long flow to a short flow is smaller than 2. Thus, each long flow achieves more than half of throughput of each short flow.

Table I shows the aggregated throughput at switches R_1 and R_2. The results are quite close to 1 Gbps with all the three protocols. In SAB, switch R_1 divides its buffer size by 15 and then assigns the result to each flow belonging to sets S and L. However, the flows in set L will pass the second bottleneck switch R_2 after R_1. Switch R_2 reassigns smaller congestion window to the passing flows. Therefore, at switch R_1, the long flows inject less traffic than that assigned by R_1. But R_1 still achieves the maximum utilization at its bottleneck link since SAB can tolerate that the injected traffic has some difference from the expected amount, which is similar to the impact caused by over-estimation of the number of flows.

3) Small delay: In the following experiments, we connect four servers to a NetFPGA switch. Three of them respectively transmit 10 long-lived flows to the fourth one. 10 seconds after the long-lived flows being generated, one sender of the three generates short flows with flow sizes ranging from 1 KB to 20 KB to the receiver in turn. By analyzing the experimental

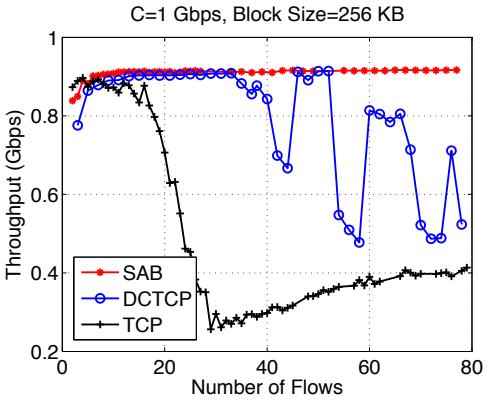


Fig. 10. Incast: Throughput of different protocols with different number of flows.

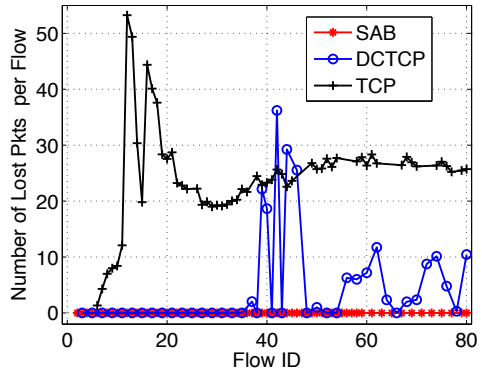


Fig. 11. Incast: Average number of lost packets per flow per second.

results, we find that all the goodput of the fourth receiver is more than 0.99 Gbps. Figure 9 depicts the completion time of the short flows with different sizes, showing that SAB performs best. Because in SAB the flows grab their fair share of bandwidth quite quickly, each flow can be finished within fewer RTTs than TCP and DCTCP. The latency of TCP is the largest, about 4 milliseconds. In DCTCP, the flows complete in about 1 milliseconds since DCTCP maintains small queue length.

4) Incast: Generally, the Incast traffic pattern happens in a rack where one server acts as the receiver and the others transmit synchronized blocks to it [11]. All the flows aggregate at the same point, namely, the rack switch. To build up this scenario using the NetFPGA switch with four ports, we let three server generate multiple flows and the fourth one acts as the receiver. The receiver requests a 256 KB block to each sender. After receiving all the blocks, the receiver requests the next blocks. The average throughput with different number of flows is drawn in Figure 10. We can readily find that TCP suffers throughput collapse after about 32 flows, which is the same as the results with 1 Gbps link capacity, 256 KB block size and 512 KB buffer size per port in [7]. DCTCP performs better than TCP does. However, after 40 connections, its throughput gradually decreases. While SAB performs well and stable.

Figure 11 depicts the average number of packet losses per flow per second with different number of flows. TCP

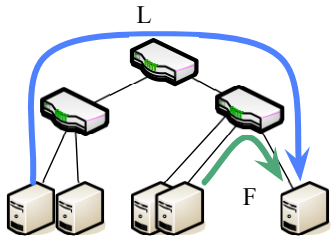


Fig. 12. Outcast: Topology.

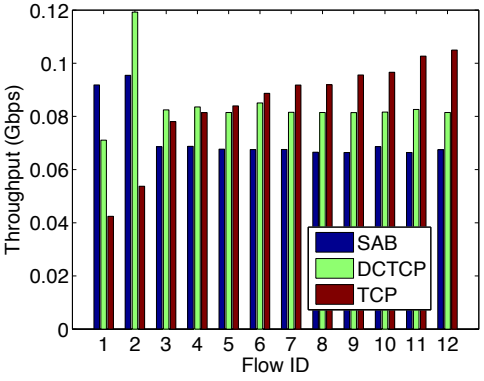


Fig. 13. Outcast: Throughput.

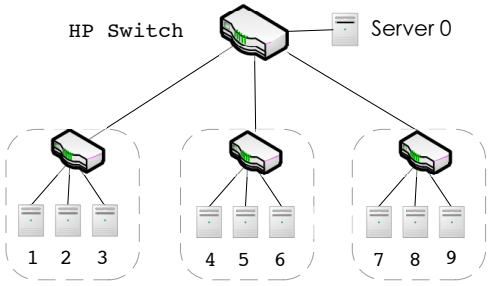


Fig. 14. The testbed topology. The switchs are NetFPGA. Server 0 is the controller of traffic generation.

loses the maximum packets. DCTCP sometimes loses many packets when the number of flows is large. This is because occasionally the end host cannot respond quickly enough to the Incast burst. While SAB does not lose any packets since the number of injected packets does not exceed the network capacity. Besides, the bandwidth share is fair and thus no one will delay the others.

5) Outcast: An Outcast scenario is built in this experiment according to the configuration in [10]. Outcast happens when the flows with different RTTs from different input ports are transmitted to the same output port and the number of flows with smaller RTTs is fewer. Generally, the flows with smaller RTTs should obtain more bandwidth. However, in the Outcast scenario, the throughput results exhibit inverse RTT bias. Figure 12 shows the topology to test the TCP Outcast problem. Denote N_f as the number of short flows F and N_l as the number of long flows L. We conduct series of experiments with different N_f and N_l. Note that when the number of flows is larger than the number of servers, one server will generate

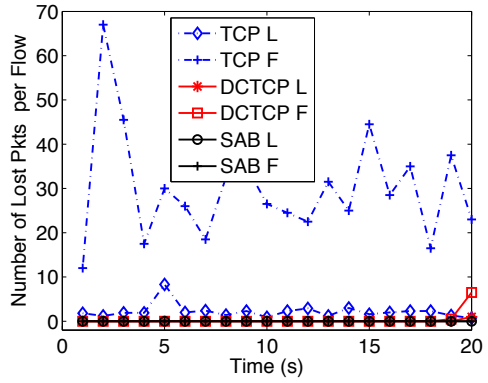


Fig. 15. Outcast: Num of lost packets per flow per second.

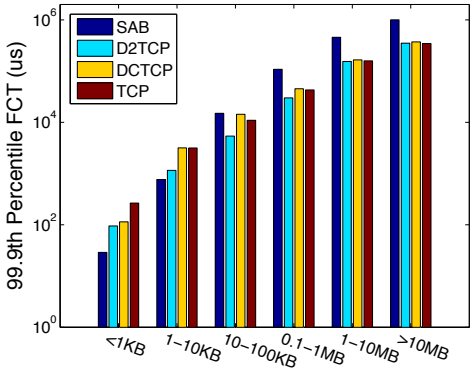


Fig. 17. 99.9th percentile completion time of background flows.

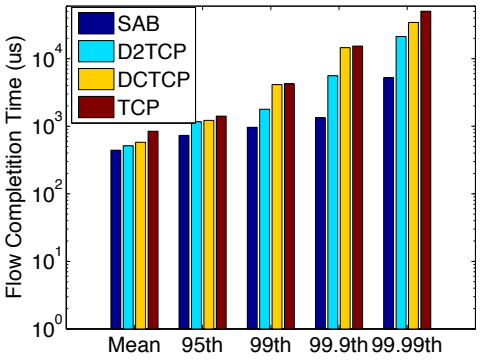


Fig. 16. Completion time of query flows.

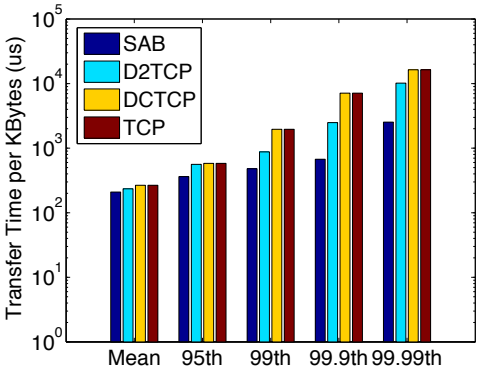


Fig. 18. Transfer time per kilobytes of data.

multiple flows. TCP indeed suffers the Outcast problem when N_l is several times of N_f, while SAB can effectively solve the problem due to SAB's feature of rare loss.

Figure 13 illustrates the throughput of different protocols with $N_f = 2$ and $N_l = 10$. Flows with ID 1 and 2 represent the short flows, and the others are the 10 long flows. Obviously, TCP suffers from the Outcast problem since Flows with ID 1 and 2 have smaller RTTs but lower throughput. Yet the flows with larger RTT get more bandwidth. SAB effectively overcomes this problem. It still exhibits its fairness as in other scenarios. As for DCTCP, the Outcast problem is also mitigated, but is not eliminated. This is partly because the marking operation in DCTCP generates port blackout as the drop operation in TCP, that is, many packets of some flows are unluckily marked while the packets of the other flows are rarely marked [10].

Figure 15 illustrates the number of lost packets. In TCP, the flows with smaller RTT lost many packets, more than the flows with larger RTT. This can explain why TCP undergoes the Outcast problem. DCTCP loses only several packets since it maintains small queue size. SAB does not lose any packets.

6) Benchmark Traffic: We generated realistic traffic, including query, short messages and background flows, based on the cumulative distribution function of the interval time between two arrival flows and the probability distribution of background flow sizes in [4]. The distribution curves are gotten based on a large amount of measured data from 6000 servers in a real data center network [4]. The size of each query message is 2 KB. The queries and short messages have deadlines that are proportional to their flow sizes. The scale factor in our experiment is 10 millisecond per kilobytes data, that is, the deadline of a 2 KB query flow is 20 milliseconds. We did experiments in a testbed as shown in Figure 14. The traffic lasts 10 minutes, and is consist of 33 K flows.

Figure 16 shows the completion time of query flows. In data centers, since the performance of parallel, delay-sensitive applications is bounded by tail latency, the tail of the flow completion time is an important metric. Therefore, we show not only the mean of the flow completion time, but also the tail values. Clearly, SAB performs much better than both DCTCP, D^{2}TCP and TCP, especially at the tail of the distribution. The reason is that the query flows can quickly reach their fair share of bandwidth, and the long flows do not form congestion at the switch buffers in SAB. D^{2}TCP works better than DCTCP because it allocates more bandwidth to short flows.

Figure 17 presents the 99.9th percentile completion time of the background flows with different flow sizes. SAB performs better than DCTCP, D^{2}TCP and TCP for flows smaller than 10KB, while it performs worse for flows larger than 10KB. This is because the short flows in SAB can quickly get their fair bandwidth, and the bandwidth taken by long flows per unit time in SAB is less than that in DCTCP, D^{2}TCP and TCP. Therefore, long flows in SAB need more time to finish, that is, with a constant network capacity, SAB decreases the performance of long flows a little to make short flows complete faster.

Now let's see how SAB performs as a whole. We computed the time required to transfer one kilobyte of data and depicted

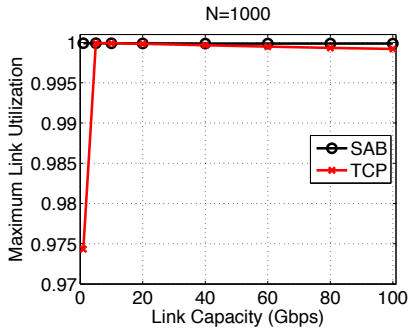


Fig. 19. Maximum link utilization of SAB and TCP.

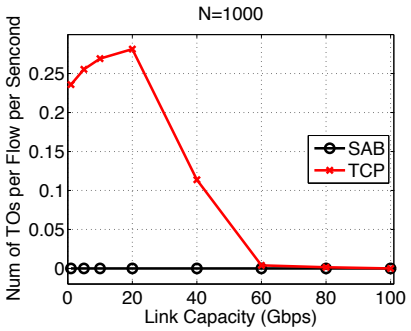


Fig. 20. TimeOut periods suffered by one flow per second.

the results in Figure 18. We can see that SAB transfers data faster than DCTCP, D^{2}TCP and TCP, that is, SAB utilizes the network bandwidth more efficiently. The 99.9th percentile value is almost the same as the mean value, which indicates that the tail of SAB is relatively short.

VI. Simulations

We conducted experiments to evaluate the performance of SAB in last section on a small-scale testbed due to the limitation of equipments. To assess whether the proposed SAB performs well in large-scale topologies, such as with higher bandwidth 10 Gbps or more servers, we implemented SAB on the ns-2 platform and evaluated its performance.

We conducted many series of simulations. Here only the results of large-scale simulations with higher link capacity and more senders are presented to show the scalability of SAB. The parameters used in each simulation are described in the corresponding subsections.

A. Impact of Capacity

With the proliferation of on-line services, data centers with higher bandwidth have gained much attention in industry [30]. Can the proposed SAB be employed in future high-speed data centers? Note that the fundamental assumption of SAB is $B > CD$, that is, the buffer size of a switch port is larger than the product of link bandwidth and round trip propagation delay. Obviously, higher link capacity enlarges CD. However, The buffer size also increases with higher link capacity. By investigating the datasheets of different switches, we found that the buffer size per port is almost positively proportional to the port capacity. Taking the switch products of Cisco and HP as examples, Cisco Catalyst 6500 Series 10G Ethernet has 16-200 MB buffer memory per port [31]. HP 6600 series, such as HP 6600-48G-4XG Switch (J9452), have about 4-9 MB buffer size per port [32]. The switch buffer size per port is greatly larger than CD. Therefore, in the data center networks with high-speed link, the condition required by SAB can still be satisfied.

To further validate it, we conducted a series of simulations with link capacities ranging from 1 Gbps to 100 Gbps. 1000 long-lived flows share a bottleneck link. Specifically, 1001 servers connect to a switch. 1000 of them act as senders, while the other one is the receiver. ε is set to 0.6. The round trip propagation delay is 100 microseconds. We take conservative buffer size, that is, the buffer size for 100 Gbps is 3MB. To conform with the positive proportional relationship between the commodity switch buffer size and the port capacity, we construct a function of the link capacity C in units of Gbps to determine the buffer size per port, $B = (300C + 3000)/11$ KB. Using this function, $B = 300$ KB when $C = 1$ Gbps, and $B = 3$ MB when $C = 100$ Gbps. Note that when conducting simulations on ns-2 with higher link capacity, the $window$ value at the receiver side needs to be enlarged so that it will not clamp the sending window.

Figure 19 depicts the aggregated throughput of SAB and TCP. We can see that SAB works well with different link capacities, and the maximum link utilization of SAB is a little higher than TCP.

Although TCP performs well in terms of the aggregated throughput. It incurs many timeouts as shown in Figure 20. If a short flow unfortunately encounters even if one timeout, its completion time will be largely increased. Timeouts do not happen in SAB in various link capacities. This is desirable to reduce the latency of short messages.

B. Impact of Rate Limitation

To evaluate the performance of SAB with rate limitation at the application layer. We use the topology shown in Figure 6 to conduct a series of simulations. We let $N_s = 300, N_l = 200, N_f = 200$. Figure 22 shows the throughput results of the flow sets without rate limitation and with rate limitation of 150 Mbps on the long flow set L. On the left, without rate limitation, the long flows can obtain about 270 Mbps bandwidth. Flow sets S and F take about 720 Mbps, respectively. Then we set the rate limitation of the flow set L to 150 Mbps. We can see that the flow sets S and F accordingly increase their throughput to about 840 Mbps, that is, the bottleneck links at the switches R_1 and R_2 are still fully utilized. This is because the flows with rate limitation inject fewer packets, which reduces the queuing delay. However, the number of packets injected by the other flows during each round trip time keeps the same. Since the throughput of a flow is inversely proportional to RTT, the throughput of the flows without rate limitation increases.

C. Impact of ε

In Section IV-C, we have stated how to set the parameter ε. To comprehensively see the impact of ε on the throughput and

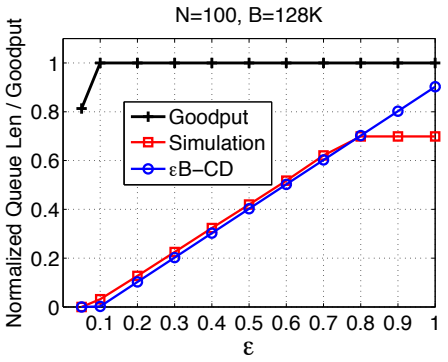


Fig. 21. Impact of ε: Goodput and queue length.

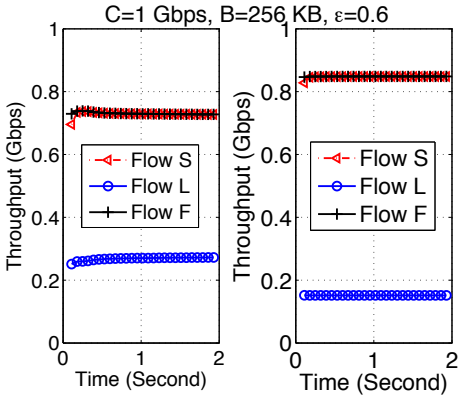


Fig. 22. Without (left) or with (right) rate limitation on flow set F.

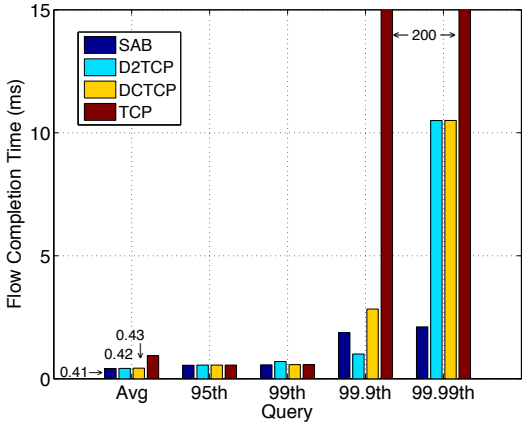


Fig. 23. Completion time of query flows in large topology.

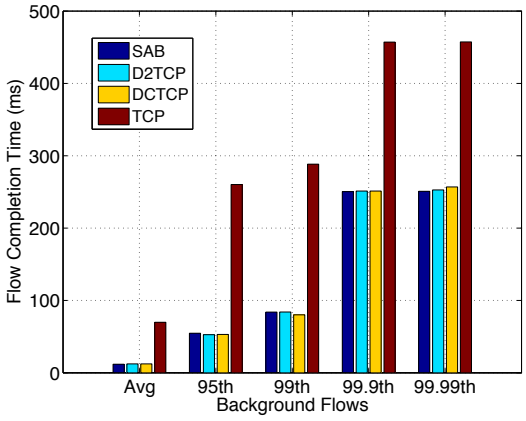


Fig. 24. Completion time of background flows in large topology.

the queue length, we let 100 servers communicate with one server with different ε. The bottleneck buffer size is $B = 128$ KB. The packet size is 1 KB. The capacity $C = 1$ Gbps and the round trip propagation delay $D = 100$ us. Figure 21 depicts the normalized goodput and queue length at the bottleneck port. We can see that when $\varepsilon \geq 0.1$, the network capacity can be fully utilized. This is because $0.1B = 12.8$ KBytes, which is greater than $CD = 12.5$ KBytes. With respect to the queue length, the values at the bottleneck switch in simulations approximately equal $\varepsilon B - CD$ except from $\varepsilon \geq 0.9$. Because when $\varepsilon < 0.9$, we have $\varepsilon B < 100$ KB. Thus, each sender's congestion window is smaller than one. Then the MSS will be reduced to be the allocated bandwidth. Therefore, the summation of injected packets equals εB and thus the queue length is close to $\varepsilon B - CD$. While when $\varepsilon \geq 0.9$, the decimal part of the congestion window is ignored, thus, the queue length in simulation is smaller than the computed value.

D. Practical Traffic

We generated practical traffic according to the traffic characteristics described in [4] on the ns-2 platform. Then we conducted simulations in a topology similar to Figure 14, but with more servers and higher link bandwidth. The simulated data center topology has 20 racks, each rack has 20 servers. The capacity of each link within a rack is 1 Gbps. The capacity of one link between a rack switch and the core switch is 10 Gbps.

Figure 23 shows the flow completion time of queries with different protocols. We can see that SAB performs much better than D^{2}TCP, DCTCP and TCP. On average, each query flow in SAB is completed 0.01 milliseconds faster than in D^{2}TCP and 0.02 milliseconds faster than in DCTCP. The benefits of SAB is more obvious in terms of the tail latency suffered by each query flow. Particularly, the 99.9th percentile tail latency in D^{2}TCP is smaller than in SAB. However, the 99.99th percentile tail latency in D^{2}TCP is much larger than its 99.9th percentile tail latency, which indicates that D^{2}TCP provides some benefits in terms of reducing the flow completion time of deadline-critical flows. However, it possibly sacrifices the performance of some other deadline-critical flows. The tail latency of SAB is quite small since each flow can obtain its fair bandwidth using only one RTT. We can see that the 99.99th percentile tail latency in SAB is almost the same as its 99.9th percentile tail latency. The tail latency of TCP is quite high. This is because TCP is a loss-driven protocol. Since the query flow size is 2KB, any packet dropping can only be recovered by timeouts. The default minimum timeout value, RTOmin, in TCP is 200 milliseconds. If an unfortunate flow encounters a timeout, the flow completion time will become quite large.

Figure 24 plots the flow completion time of background flows with different protocols. SAB, D^{2}TCP and TCP perform almost the same. The 95th percentile tail latency of SAB is a little larger than D^{2}TCP and DCTCP since the query flows in SAB take more bandwidth than in D^{2}TCP and DCTCP. TCP performs worst due to many timeout periods.

VII. Conclusion

Data center networks have an unique feature that the buffer size is larger than the BDP. The proposed transport protocol in this paper, called SAB, is a simple yet effective algorithm exploiting the benefits of this feature. In SAB, switches determine the congestion window of each flow by allocating their buffer space. Since the buffer size is larger than the BDP, the bottleneck capacity can be fully utilized. SAB converges fast because the end hosts can obtain their fair share of bandwidth in only one RTT. Besides, SAB rarely loses packets because the number of injected packets is smaller than the network capacity and MSS will be reduced when the congestion window is smaller than one. SAB is implemented on a NetFPGA-based testbed as well as the ns-2 platform. The experiment and simulation results demonstrate that SAB indeed converges fast and does not suffer from the TCP Incast and Outcast problems due to its property of rare loss.

Appendix

A. Proof of PROPOSITION 2.

Proof. Since the window size of each flow is set to $\frac{\varepsilon B}{N}$, the summation of the congestion window size of all the flows passing switch r is $\sum_{i=1}^{N} W_i = \frac{\varepsilon B}{N} N = \varepsilon B \frac{1}{1 \pm \eta}$.

According to eq. (1), to fully utilize the bottleneck link, the summation of the windows of all the flows should satisfy $CD_{max} \leq \frac{\varepsilon B}{1 \pm \eta} \leq B + CD_{min}$. Therefore, we can get that $\varepsilon B \in [(1+\eta)CD_{max}, (1-\eta)(B+CD_{min})]$. Thus, we have

$$(1+\eta)\frac{CD_{max}}{B} \leq \varepsilon \leq \min\{1, (1-\eta)\frac{B+CD_{min}}{B}\} \quad (6)$$

Because $(1+\eta)CD_{max} \leq (1-\eta)(B+CD_{min})$, we obtain that the maximum allowed estimation error is

$$\eta_{max} = \frac{B + CD_{min} - CD_{max}}{B + CD_{min} + CD_{max}} \quad (7)$$

B. Proof of PROPOSITION 3.

Proof. Since $N_s < N_f$, the long flows inject fewer packets than that allocated by switch R_1. According to PROPOSITION 1, as long as the number of packets that arrive to R_1 is larger than CD_{max}, that is,

$$\varepsilon B(\frac{N_s}{N_s + N_l} + \frac{N_l}{N_l + N_f}) > CD_{max} \quad (8)$$

the link associated with the switch R_1 can be fully utilized. Since the estimation error tolerance is η. a and b should satisfy $\frac{1}{1+a} + \frac{b}{1+b} > \frac{1}{1+\eta}$.

Acknowledgement

The authors gratefully acknowledge the anonymous reviewers for their constructive comments as well as Aurojit Panda, Peng Zhang, and Hongkun Yang for polishing the manuscript. This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61225011, National Basic Research Program of China (973 Program) under Grant No.2012CB315803 and 2009CB320504.

References

[1] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson1, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *ACM SIGCOMM*, pp. 303–314, Aug.2009.

[2] Y. Chen, R. Griffith, J. Liu, R. Katz, and A. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *the 1st ACM workshop on Research on enterprise networking*, pp. 73–82, 2009.

[3] S. Radhakrishnan, Y. Cheng, J. Chu, and A. Jain, "TCP Fast Open," in *ACM CoNEXT*, 2011.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, and J. Padhye, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, pp. 63–74, 2010.

[5] A. Shieh, S. Kandula, and A. Greenberg, "Seawall: Performance Isolation for Cloud Datacenter Networks," in *USENIX workshop on HotCloud*, 2010.

[6] J. Dean, S. Ghemawat, and G. Inc, "MapReduce: Simplified Data Processing on Large Clusters," in *USENIX OSDI*, 2004.

[7] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *USENIX FAST*, 2008.

[8] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, N. Sutin, A. Agarwal, T. Herbert, and A. Jain, "An Argument for Increasing TCP's Initial Congestion Window," *ACM SIGCOMM CCR*, vol. 40, no. 3, pp. 27–33, 2010.

[9] J. Zhang, F. Ren, and C. Lin, "Modeling and Understanding TCP Incast in Data Center Networks," in *IEEE INFOCOM*, pp. 1377–1385, 2011.

[10] P. Prakash, Y. C. Hu, R. Kompella, and A. Dixit, "The TCP Outcast Problem : Exposing Unfairness in Data Center Networks," in *USENIX NSDI*, 2012.

[11] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in Data Center Networks," in *ACM CoNEXT*, 2010.

[12] S.Floyd and V.Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Netw.*, 1993.

[13] P. Mckenney, "Stochastic Fairness Queueing," in *IEEE INFOCOM*, pp. 733–740, 1990.

[14] N. Kung and R. Morris, "Credit-based Flow Control for ATM Networks," *IEEE Network*, vol. 9, no. 2, pp. 40–48, 1995.

[15] J.Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM*, pp. 303–314, Sep.1998.

[16] H. T. Kung and K. Chang, "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks VC2," in *IEEE INFOCOM*, pp. 239–252, 1995.

[17] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-Aware Datacenter TCP (D2TCP)," in *ACM SIGCOMM*, 2012.

[18] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *USENIX NSDI*, 2011.

[19] R. W. Watson, "Timer-Based Mechanisms in Reliable Transport Protocol Connection Management," *Computer Networks*, 1981.

[20] R. Braden, "RFC1644: T/TCP–TCP Extensions for Transactions Functional Specification," 1994.

[21] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-delay Product Networks," in *ACM SIGCOMM*, 2002.

[22] N. Dukkipati and N. Mckeown, *Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly*. Stanford University, 2008.

[23] C. Wilson and T. Karagiannis, "Better Never than Late : Meeting Deadlines in Datacenter Networks," in *ACM SIGCOMM*, pp. 50–61, 2011.

[24] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, pp. 733–740, 1988.

[25] D. Abts and B. Felderman, "A Guided Tour through Data-center Networking," *ACM Queue*, vol. 10, no. 5, p. 10, 2012.

[26] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High-speed Links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, 2006.

[27] N. Duffield, C. Lund, and M. Thorup, "Properties and Prediction of Flow Statistics from Sampled Packet Streams," in *the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pp. 159–171, 2002.

[28] N. Parvez, A. Mahanti, and C. Williamson, "An Analytic Throughput Model for TCP NewReno," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 448–461, 2010.

[29] http://www.stanford.edu/ alizade/Site/DCTCP.html.

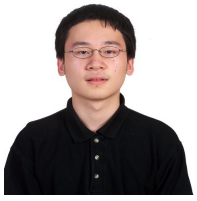
[30] "IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force." http://www.ieee802.org/3/ba/index.html.

[31] "Cisco Catalyst 6500 Series 10 Gigabit Ethernet Interface Modules." http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/product_data_sheet09186a00801dce34.html.
[32] "HP ProCurve 6600 Switch Series." http://www.hp.com/rnd/pdfs/datasheets/HP_ProCurve_Switch_6600_Series.pdf.

Jiao Zhang is currently a Ph.D candidate of the Department of Computer Science and Technology, Tsinghua University, China. Her supervisor is Prof. Fengyuan Ren. Since Aug. 2012, she has been a visiting student in the networking group of ICSI, University of California, Berkeley. Her recent research mainly focuses on traffic management in data center networks. She also has done some work on data aggregation and energy-efficient routing in wireless sensor networks before.

Fengyuan Ren is a professor of the Department of Computer Science and Technology at Tsinghua University, Beijing, China. He received his B.A and M.Sc. degrees in Automatic Control from Northwestern Polytechnic University, China, in 1993 and 1996 respectively. In Dec. 1999, he obtained Ph.D degree in Computer Science from Northwestern Polytechnic University. From 2000 to 2001, he worked at Electronic Engineering Department of Tsinghua University as a post doctoral researcher. In Jan. 2002, he moved to the Computer Science and Technology Department of Tsinghua University. His research interests include network traffic management and control, control in/over computer networks, wireless networks and wireless sensor networks. He (co)-authored and co-authored more than 80 international journal and conference papers. He is a member of the IEEE, and has served as a technical program committee member and local arrangement chair for various IEEE and ACM international conferences.

Xin Yue is currently a Ph.D candidate majoring in EE at Tsinghua University, under the advising of Prof. Xing Li. Xin Yue's research interests are primarily centered on the fields of data center networks and IPv6 transition mechanisms. Besides, he's also an advocate of open source and creative commons.



Ran Shu is pursuing his Master degree in the Department of Computer Science and Technology of Tsinghua University, Beijing, China. He received his B.E. degree in Computer Science and Technology from Tsinghua University, Beijing, China in 2011. He is supervised by professor Fengyuan Ren and professor Chuang Lin now. His research interests include congestion control and data center networks.

Chuang Lin is a professor of the Department of Computer Science and Technology at Tsinghua University, Beijing, China. He is an Honorary Visiting Professor, University of Bradford, UK. He received the Ph.D. degree in Computer Science from Tsinghua University, China in 1994. His current research interests include computer networks, performance evaluation, network security analysis, and Petri net theory and its applications. He has published more than 300 papers in research journals and IEEE conference proceedings in these areas and has published four books. He is a senior member of the IEEE and the Chinese Delegate in TC6 of IFIP. He served as the Technical Program Vice Chair, the 10th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004); the General Chair, ACM SIGCOMM Asia workshop 2005 and the 2010 IEEE International Workshop on Quality of Service (IWQoS 2010). He is an Associate Editor of IEEE Transactions on Vehicular Technology and an Area Editor of Computer Networks and the Journal of Parallel and Distributed Computing.