TFC: Token Flow Control in Data Center Networks

Jiao Zhang[†§], Fengyuan Ren[†‡], Ran Shu[†‡], Peng Cheng[†♭]
[†]Department of Computer Science and Technology, Tsinghua University
[‡]Tsinghua National Laboratory for Information Science and Technology
[§]State Key Laboratory of Networking and Switching Technology, BUPT
[♭]Microsoft Research Asia
jiaozhang@bupt.edu.cn, {renfy, shuran}@csnet1.cs.tsinghua.edu.cn, pengc@microsoft.com

Abstract

Services in modern data center networks pose growing performance demands. However, the widely existed special traffic patterns, such as micro-burst, highly concurrent flows, on-off pattern of flow transmission, exacerbate the performance of transport protocols. In this work, an clean-slate explicit transport control mechanism, called Token Flow Control (TFC), is proposed for data center networks to achieve high link utilization, ultra-low latency, fast convergence, and rare packets dropping. TFC uses *tokens* to represent the link bandwidth resource and define the concept of *effective flows* to stand for consumers. The total tokens will be explicitly allocated to each consumer every time slot. TFC excludes in-network buffer space from the flow pipeline and thus achieves zero-queueing. Besides, a packet delay function is added at switches to prevent packets dropping with highly concurrent flows. The performance of TFC is evaluated using both experiments on a small real testbed and large-scale simulations. The results show that TFC achieves high throughput, fast convergence, near zero-queuing and rare packets loss in various scenarios.

Categories and Subject Descriptors C.2.2 [*Computer-Communication Networks*]: Network Protocols–TCP/IP

Keywords Data Centers, Flow Control, Low Latency, Rare Loss, Fast Convergence

1. Introduction

The performance demand of services is growing in modern data center networks. For example, streaming computing like Storm [4] supporting online data processing is more sensitive to average and tail latency than batch models like MapReduce for offline processing. In memcached systems [18], zero packets loss is a key performance metric since the retransmission after loss severely impacts the transmission performance.

However, the special traffic characteristics, such as short traffic bursts [13, 22], highly concurrent transport protocols [2, 12], on-off pattern of flow transmission [34], significantly deteriorate the performance of flows. Traffic bursts easily lead to severe congestion and packets dropping, which will result in long flow completion time [15]. Highly concurrent flows might cause network congestion even if the congestion window of each flow is only one Maximum Segment Size (MSS). Many transport protocols designed for data centers could not deal with these issues properly [7, 35]. The on-off pattern of flow transmission possibly causes wastage of allocated bandwidth [37].

What are the desirable properties for an applicable data center transport protocol? First, *fast convergence.* A short flow usually consists of only several packets. The sluggishness of a congestion window evolution algorithm will postpone the transmission of short flows. Besides, the on-off transmission pattern of flows also needs transport protocols quickly response to dynamic loads. Second, *zero packet loss.* Packets dropping possibly result in a series of problems, such as TCP Incast [36], TCP Outcast [32], long query completion time [7]. Avoiding packets dropping can effectively solve these problems [15]. Third, *low latency.* Keeping zero-queueing can reduce the network latency caused by congestion to the extreme extent.

Most of the proposed transport protocols for data centers could only satisfy some of the desirable properties described above. For example, the congestion window adjustment mechanism involved in TCP and its variants (i.e. DCTCP) incline to aggressively probe the available bandwidth through injecting excessive packets to networks, which results in persistent queue backlog and thus fundamentally

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, contact the Owner/Author(s). Request permissions from permissions@acm.org or Publications Dept., ACM, Inc., fax +1 (212) 869-0481.
EuroSys '16, April 18 - 21, 2016, London, United Kingdom
Copyright © 2016 held by owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4240-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2901318.2901336

limits latency improvements. The explicit rate-based congestion control protocols, such as D^3 [37], cannot deal with silent flows, i.e. flows are held not closed and wait for data to resume. To better adapt to these intermittent flows, application modifications are inevitable. In Fastpass [31], the scalability is restricted by the capacity of the centralized controller which limits its deployment.

To satisfy all above-mentioned properties in data center networks, the injected traffic in total by end hosts should fill the network bottleneck link but eliminate in-network persistent queue backlogs.

In this paper, a novel window-based transport control protocol is proposed for data center networks, which is called Token Flow Control (TFC). Two important concepts, *Token* and the *Number of effective flows*, are defined in TFC. Specifically, Token represents the amount of traffic that can be transmitted by a link in a time slot. The number of effective flows stands for how many full windows of data packets will be injected by all the flows in a time slot. By transforming the link transmission capacity to tokens and dynamically allocating these tokens to passing flows, TFC excludes the buffer space from the flow pipeline and thus prevents building up persistent queues. If we could get the accurate value of tokens and the number of effective flows in a time slot, then we could allocate the total tokens to flows according to any allocation policies.

To achieve both fast convergence and accurate congestion window computation, the time interval of a time slot is finally set to the Round Trip Time (RTT) of a flow. Besides, to achieve zero queueing delay, the RTT used to compute the token value and the RTT used to count the number of effective flows are decoupled. End hosts mark the first sent data packet during each round to facilitate switches to get the two kinds of RTT values.

What's more, to prevent the performance deficiencies under the micro-burst traffic pattern, a window acquisition phase is added after the flow establishment phase. And to avoid buffer overwhelming when the number of active flows is too large, if the computed congestion window is smaller than one MSS, packets will be delayed at switches .

The advantages of the proposed TFC mechanism are mainly threefold. 1) Ultra-low latency. Since buffer is excluded from the flow pipeline, TFC can achieve near zero queueing delay by guaranteeing the summation of the injected traffic by end hosts not exceeding the capacity of the pipeline. Besides, each flow could quickly converge to its proper congestion window after two RTTs. Therefore, the flow completion time could be decreased dramatically. 2) Achieving near-zero loss. By deliberately designing the window acquisition phase and packet delay function at switches, TFC could avoid packets loss caused by bursty traffic and large-scale concurrent flows in data center networks. 3) Switches do not need to maintain states for each flow and the transport control function at end hosts become more simple.

The performance of TFC is evaluated in our testbed with Dell end hosts and NetFPGA [1] switches. The experimental results show that TFC guarantees high throughput, near zero queuing, fast convergence and rare packet loss. The window acquisition phase and the packet delay function at switches enable TFC to work well with burst traffic and when the number of active flows is rather large. Simulation results on the ns-2 platform also indicate that TFC exhibits good scalability.

2. Design Space

Different from traditional networks, services in data center networks have many special characteristics, which results in new requirements for transport protocols besides high link utilization.

Fast Convergence. In data center networks, about 90 percent of flows are quite short [5]. A short flow is generally consisted of only several packets. To reduce the completion time, transport protocols should enable flows to quickly converge to a proper share of bandwidth. TCP and its variants let flows evolve to their fair share from a quite small initial congestion window, which is not responsive enough to this kind of flows and causes them not to obtain their fair bandwidth share before ending.

Furthermore, a flow could keep silent during some time after establishment. For example, in the Storm computing framework, a TCP connection exchanges messages for several executors [34]. It is likely that the connection will transmit data intermittently. To keep full link utilization, the bandwidth occupied by silent flows should be taken up by other active flows as quickly as possible. TCP or TCP variants fail to work well in this kind of scenarios due to slow convergence. Although some explicit transport protocols proposed for data center networks, such as D^3 [37], could enable flows to quickly converge to a proper rate, they rely on SYN and FIN to count flows. The silent flow will cause some bandwidth wastage.

Zero Packet Loss. In large-scale data center networks, TCP suffers from a growing number of performance issues, including TCP Incast [14, 39], TCP outcast [32], long query completion time [37], out-of-order, etc.. Avoiding packets dropping or providing quick packet loss notification can effectively solve or alleviate these problems [15]. Many recently proposed transport protocols can significantly reduce the number of dropped packets by limiting the queuing length [7, 35] or controlling the sending rate [21, 37]. However, most of them could not work well in scenarios with highly concurrent flows. For example, in DCTCP, the experimental results show that when the number of senders is so large that a sender's congestion window is smaller than two packets in the Incast communication pattern, some flows will suffer packets dropping or even timeouts [7]. Highly

concurrent flows widely exist in today's data centers. Lots of services in data center networks are completed by a large number of cooperated servers based on the parallel computing frameworks, such as MapReduce [16] or Storm [4]. For example, both Google and Microsoft report that each interactive web-search service consists of 10s-1000s of servers on average [12, 35]. Thus, many servers will synchronously transmit data to the same server in data center networks. Therefore, *it is desirable to design a transport protocol with zero packets dropping even there are massive concurrent flows.*

Low Latency. Large latency brings great negative impact on the profits of cloud service providers. Microsecond computing has been identified as a great challenge that hampers the development of highly responsive, massively scaled data centers [3]. Many cloud services are conducted using distributed realtime computation systems, such as Storm [4]. Network latency is a substantial component of the coordination delay in the computing frameworks. Besides, in interactive applications such as web services, a HTTP request will generally trigger several hundreds of internal requests in data centers. For example, a Facebook HTTP request will trigger an average of 130 internal requests inside the Facebook site [10]. The internal requests are sequentially dependent, which causes a large cumulative latency. Thus, reducing the network latency is critical to decrease the response time of services. Network end-to-end latency could be reduced from different perspectives, including packet processing delays in the OS stack, network interface card and network switches, and delays caused by network congestion [29]. In data center networks, the round trip delay is extremely low, typically on the order of a few hundred microseconds [36]. In contrast, queueing delay could be quite large [7]. Thus, *keeping zero-queueing is a desirable requirement of transport protocols to reduce the network congestion delay to the extreme extent in data center networks.*

3. Basic Idea and Challenges

3.1 Basic Idea

Broadly, the existing transport control protocols could be classified into window-based or rate-based approaches. Window-based solutions offer many advantages, such as data-driven clocking and inherent stability [33]. In contrast, rate-based protocols are harder to be implemented since correctly using timer to control rate is non-trivial [33]. From another perspective, current transport protocols could also be classified into explicitly allocating bandwidth by network devices or implicitly probing available bandwidth at end hosts. However, due to limited information at each host, implicitly probing bandwidth hardly meets the goals of fast convergence, zero-queueing and rare packets loss. Therefore, we will design an *explicit window-based transport control* protocol for data center networks in this work.

There are some typical explicit window-based transport protocols [17, 23]. Unfortunately, the convergence rate of them are not fast enough [40] since they rely on window evolution round by round to achieve fairness and efficiency. Also, they hardly achieve zero-queueing and rare packets loss in various scenarios since the evolved window feedback is possibly not accurate. In history, there is a well-known credit-based flow control mechanism designed for ATM networks [26], which can achieve rare packets dropping by strictly controlling that a receiver can notify a sender to transmit a certain amount of traffic only when the receiver has the corresponding space (credit) to accommodate the traffic. However, it could not achieve zero buffering.

Enlightened by the previous explicit transport protocols and the credit-based flow control mechanism, we aim to design a transport control protocol that satisfies the above three goals by obtaining the flow pipeline capacity without buffer space (*credit*) and then *explicitly* allocating the capacity to passing flows. In this way, first, since the total credit value to be allocated does not include the buffer space, there will be *no queuing delay*. Second, since each sender will inject traffic strictly according to the obtained credit value and no more traffic will be gradually increased like implicit transport protocols, *rare packets dropping* can be achieved. Third, if a switch could obtain the exact flow pipeline capacity without buffer space, then a flow will get its proper window in one round. Thus, *fast convergence* can be satisfied.

However, how to exactly measure the flow pipeline capacity without buffer space, the number of passing active flows, and how to deal with the work-conserving problem and highly concurrent flows are extremely challenging.

3.2 Challenges

Excluding buffer space from the flow pipeline capacity. Implicitly probing appropriate window size for flows, like traditional TCP, usually treats the buffer space as part of the flow pipeline. More and more packets will be injected to the network until some packets are dropped or the queue length exceeds a threshold. To guarantee zero-queueing and high link utilization, the ideal congestion window size is $c \times RTT$ and the buffer only needs to absorb the bursts within a round. However, in practice the buffer space is usually an order larger than $c \times RTT$. The packets in buffer only increase the round trip time which harms the responsiveness of congestion control. Thus, the flow pipeline capacity without buffer space is the resource that we want to allocate to each sender. HULL [8] probes the flow pipeline without buffer space by constructing phantom queues. However, it sacrifices about 10% of bandwidth. Since most switches along a path has their own buffer space, *without the whole path information*, it is challenging to exactly know the capacity of the flow pipeline without buffer space. What's more, the flow pipeline capacity varies for flows with different RTTs. Thus, for a switch that accommodates *different kinds of passing*

flows, learning the flow pipeline capacity without buffer space is also challenging.

Accurately estimating the number of flows. Estimating the number of flows by counting the flow handshaking messages [37] will cause *cumulative errors*, e.g., once a SYN packet is lost, then the retransmission will cause a switch count twice in the path before the packet loss location. Besides, *the flows that do not send messages could not be excluded*. Another existing method estimates the number of flows according to historical information, such as using the result of the link capacity dividing the allocated flow rate in last time slot. However, under this method, only the bottleneck switch can obtain the correct number of flows. As a result, the convergence to efficiency process is slow [40]. Therefore, we need to design a novel method to accurately estimate the number of active flows during each round.

Work-conserving. Two main scenarios will cause the work-conserving problem. First, at a switch, *a flow injects less traffic than the allocated value* by the switch since the flow obtains less bandwidth at another switch. This is common in topologies with multiple bottlenecks. Second, *some flows will not send any traffic* even if they are allocated some bandwidth. Thus, the bandwidth allocated to these silent flows should be taken back and allocated to other active flows. How to adapt to these scenarios that will cause the work-conserving problem and do not waste bandwidth is challenging.

Window of less than one. Due to the small round trip propagation delay and *highly concurrent flows* in data center networks, it is likely that congestion will happen even if each flow only injects one MSS sized packet. Many existing transport protocols could not well deal with this problem [7, 37]. Thus, it is challenging to avoid dramatical packets dropping when the congestion window of one flow becomes less than one MSS.

4. Design of TFC

In this section, we will first describe a basic abstract model, then present how to design TFC based on the basic abstract model to achieve high link utilization, fast convergence, zero-queueing and rare packets loss.

4.1 Basic Model

In TFC, we define two concepts, *Token* and the *Number of Effective Flows*, to represent the total resource and consumers in a time slot, respectively. The time slot period t is set to an arbitrary value here, and the proper value will be discussed in 4.3.

Token $T[n]$**.** $T[n]$ *represents how many data can be transmitted by a link in time slot* $n (n = 0, 1, ...)$. $T[n]$ can be computed as $c \times t$, where c is the link bandwidth and t is the duration of a time slot. Token represents the bandwidth resource provided by a link in a time slot.

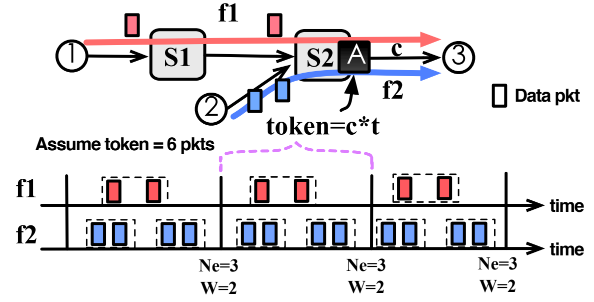


Figure 1: Illustrating the basic model of TFC. Two flows, f_1 and f_2, passing port A. Assume $rtt_1 = 2 \times rtt_2$ and the token value is 6 packets. Since there are three effective flows in a time slot, the congestion window is 2 packets for each flow.

Number of effective flows $E[n]$**.** $E[n]$ *stands for the number of full windows of data packets injected by all the passing flows in time slot* n. Effective flows can be considered as the *consumer of the bandwidth resource*. Effective flows are active. Considering that some flows possibly do not transmit any data during some time, inactive flows need to be excluded when counting the number of consumers. For example, many flows transmit data intermittently in Storm system [34]. Thus, during the silent time, the flows are not effective.

We only consider the basic objective of fair bandwidth share with RTT bias, that is, we allocate an equal window to every flow passing the same port of a switch. Then the number of consumers should equal the number of the rounds of all the flows in a time slot. Next, we will show how TFC just allocates tokens to all senders no matter how long a time slot is. Formally, denote f as the ID of arrival flows in time slot n and rtt_f as the RTT of flow f, then we have

$$E[n] = \sum_f \frac{t}{rtt_f} \tag{1}$$

Once we have obtained the token value, $T[n]$, and the number of effective flows, $E[n]$, we can compute the congestion window of each flow during next time slot

$$W[n+1] = \frac{T[n]}{E[n]} \tag{2}$$

Figure 1 gives a simple example to illustrate the basic idea of TFC. At port A of switch S_2, the passing flows include f_1 and f_2. The link bandwidth connecting to port A is c. Assume the round trip time of f_1 is twice of that of f_2, that is, $rtt_1 = 2 \times rtt_2$. Besides, assume the time interval t equals rtt_1 and the tokens equal $c \times t = 6$ packets. During a time slot, flow f_1 injects 1 full window of data packets and flow f_2 injects 2 full windows of data packets. Thus, the number of effective flows equals 3 and the congestion window is 2. The tokens can be exactly exhausted.

The basic model guarantees *fast convergence* since the number of effective flows is updated every time slot and each active flow can obtain its proper congestion window after one time slot. Besides, since the total arrival rates of all the flows in time slot $(n+1)$ is $\sum_f \frac{W[n+1]}{rtt_f[n+1]} = \sum_f \frac{1}{rtt_f[n+1]} \frac{c}{\sum_i \frac{1}{rtt_i[n]}}$. Thus, as long as the RTT of each flow keeps stable in time slot n and slot $(n+1)$, then the total injected traffic rate equals the link capacity c, which ensures *full link utilization*.

4.2 Measuring the Number of Effective Flows

In TFC, switches explicitly assign congestion window for each passing flow. The minimum congestion window along the path of a flow will be carried back to the sender by ACKs. [1]. According to Eq. (2), switches need to determine the accurate number of effective flows in a time slot to compute the congestion window, that is, switches need to know the number of full windows of data packets injected by all the flows in a time slot.

There are several possible methods. First, each flow sends a special packet during each round. Switches measure the number of special packets in a time slot to acquire the number of effective flows. However, this method brings large communication overhead. For example, if each round lasts for about 100 microseconds, 50 flows pass a switch and each special packet takes 64 Bytes (the minimum Ethernet frame size), then the bandwidth taken by the special packets along a path with 1 Gbps rate is about $50 \times \frac{64\times 8}{\text{1Gbps}\times\text{100us}} = 25.6\%$. Second, recording the total arrival traffic amount in time slot n, $A[n]$. The number of effective flows in time slot n, $E[n]$, can be computed as $\frac{A[n]}{W[n-1]}$. However, the congestion window of a flow equals the minimal congestion window along the path. Thus, only the bottleneck switch maintains correct congestion window of last time slot $W[n-1]$ and could get accurate number of effective flows in this way.

In TFC, we combine the advantage of above two methods. That is, each sender marks one specific packet during each round instead of sending an extra packet. Then switches could obtain the number of effective flows by counting the number of marked packets. By this method, *no per-flow state is needed to get an accurate E in TFC* and there is no wasted bandwidth for the measuring.

As shown in Figure 2, in time slot $n-1$, flow f_2 injects two windows of data packets, and flow f_1 is being established. According to the number of the marked data packets and marked SYN packet, we can infer that the number of effective flows $E[n-1] = 3$. According to the duration of one time slot, t, and the link capacity c, we can compute the tokens in a time slot. Suppose $T[n-1] = 6$ packets, then the congestion window during next time slot $W[n] = 2$ packets. Similarly, in time slot n, the number of effective flows is still $E[n] = 3$, thus $W[n+1] = 2$ packets. However, in time slot $(n+1)$, flow f_1 does not send data packets due to some reasons, such as the flow has finished or the corresponding application does not have data to send. Thus, $E[n+1]$ becomes 2, and the congestion window is 3 packets.

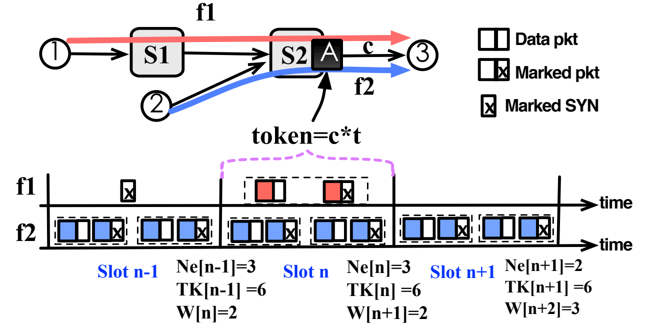


Figure 2: Framework of TFC. Senders mark the first packet of each full window of data packets to facilitate switches to measure E.

The method of determining $E[n]$ in TFC has several advantages. First, it excludes inactive flows. In practice, it is possible that some flows are inactive for a while intermittently after establishment. Using marked packets to count the number of effective flows can ensure switches quickly know the flows variations and compute the proper congestion window size. Second, it avoids cumulative errors. Using handshaking packets is a simple method of counting the number of flows [37]. However, if a SYN packet is lost after it is counted, the retransmission will cause accumulated counting error of flow number. In our method, the number of effective flows in time slot n is irrelative to the previous measured value. Thus, even if the number of effective flows has some deviations in a time slot, accurate number of effective flows still could be obtained in afterwards time slots.

4.3 Duration of a Time Slot

Theoretically, the duration of a time slot, t, in the basic model of computing the congestion window could be any values. Besides, the duration of different time slots could change. However, in practice, we need to determine a proper value to allow end users quickly converge to their appropriate bandwidth. Also, the duration of a time slot should not affect the accuracy of counting the number of effective flows in a time slot.

On one hand, the duration of one time slot should not be too large. Otherwise, once some new flows arrive, or some existing flows finish or become inactive, switches could not update the congestion window according to the new number of effective flows. Thus, TFC could not achieve the requirement of fast convergence.

On the other hand, the duration of one time slot should not be too small. Otherwise, the number of effective flows will

[1] This may cause work-conserving problem which existing schemes like [37] also have to solve. We solve it by the token adjustment mechanism described in 4.5

fluctuate dramatically. For example, in the scenario shown in Figure 1, if the duration of a time slot $t = rtt_1$, then the measured number of effective flows is 3, which equals the theoretical result $\frac{t}{rtt_1} + \frac{t}{rtt_2} = 3$. If the duration of a time slot $t = rtt_2$, then the measured number of effective flows becomes 1 and 2 alternatively. The average of the measured values of two adjacent time slots equals the theoretical result $\frac{t}{rtt_1} + \frac{t}{rtt_2} = 1.5$. However, if the duration of one time slot is smaller than rtt_2, then during some time slots, the number of effective flows will vary among 0, 1, and 2. If the measured number of effective flow is 0, then we could not compute the congestion window value.

Thus, to make a tradeoff between fast convergence and accuracy of estimating the number of effective flows, the duration of a time slot is set to one RTT of a flow. In TFC, the marked packets in each round make switches could easily measure the RTT of any flows. A switch only needs to measure the interval of two marked packets belonging to the same flow to obtain the RTT value of the flow.

Then which flow should be chosen to measure the duration of a time slot? Choosing the flows with smaller RTT lead to quicker response to flow variation, while choosing the flows with larger RTT results in more accurate measured number of effective flows. To answer this question, we first investigate the RTT values of different kinds of flows in data center networks.

The typical topologies in data center networks are 3-layer, multi-rooted trees with single or multiple paths between two end servers. In the tree-based topologies [6, 19], generally there are at most three kinds of connections between a pair of servers. One is intra-rack connections, one is cross-rack connections passing core switches, and the last one is cross-rack connections only passing edge and aggregation switches, but not passing core switches. The RTT value of a flow is mainly consisted of processing delay at end hosts, link propagation delay, and store-and-forward delay at switches. Thus, the maximum round trip time of a flow is at most three times of the minimum value in data center networks, even if RTT is dominated by transmission delays. [2] This indicates that the impact of using which flow's RTT on the performance of TFC is quite small. Besides, if a specific kind of flows is chosen, the implementation of switches will become more complex since they have to identify different kinds of flows and measure the RTT of the specific kind of flows. Also, it is possible that not all the kinds of flows will pass a switch. Based on these reasons, the duration of a time slot is determined as the RTT of any flow in TFC. The selected flow to indicate the duration of a time slot is called the ***delimiter flow***.

[2] Actually, the maximum RTT is about twice of the minimum value in our testbed.

4.4 Decoupling $E[n]$ and $T[n]$

The obtained RTT value by measuring the interval of two marked packets is related with the queueing delay. However, to keep zero-queueing, the queueing delay should be eliminated. Otherwise, larger RTT leads to larger token value, while the number of effective flows during the measured RTT does not change. Correspondingly, the congestion window of each flow increases, which will further increase the queue buildup. Thus, *the duration of a time slot used to compute the tokens $T[n]$ and the interval of measuring the number of effective flows $E[n]$ should be decoupled.* The RTT of a flow without queueing delay should be used to compute $T[n]$, while the instantaneous RTT of a flow should be used to count the number of effective flows $E[n]$.

Then how to get the RTT of a flow without queueing delay? Each switch can only know whether itself has queueing delay, but it could not know the queue length of other switches along the path. Thus, switches at TFC use the minimum of the measured RTT values as the RTT of a flow without queueing delay. Note that in store-and-forward switches, the measured RTT value may be different for packets with different size. Thus, only the marked packets with frame length larger than 1500 Bytes are used to measure RTT in TFC.

Let rtt_m^i represent the measured instantaneous RTT value of flow i. Besides, rtt_m and rtt_b stand for the instantaneous RTT value and the minimum measured RTT value of the *delimiter flow* in TFC. Then the congestion window algorithm of TFC becomes

$$T[n] = c \times rtt_b[n] \tag{3}$$

$$E[n] = \sum_i \frac{rtt_m[n]}{rtt_m^i[n]} \tag{4}$$

$$W[n+1] = \frac{c \times rtt_b[n]}{E[n]} \tag{5}$$

In this way, the summation of the arrival rate of all flows becomes

$$\begin{aligned} \sum_i r_i[n+1] &= \sum_i \frac{W_i[n+1]}{rtt_m^i[n+1]} \\ &= \sum_i \frac{1}{rtt_m^i[n+1]} \frac{c \times \frac{rtt_b[n]}{rtt_m[n]}}{\sum_j \frac{1}{rtt_m^j[n]}} \end{aligned} \tag{6}$$

In statistics $\sum_i \frac{1}{rtt_m^i[n+1]} = \sum_j \frac{1}{rtt_m^j[n]}$. On the other hand, $rtt_b[n] \leq rtt_m[n]$ is always true. Consequently, the total arrival rate of all flows do not exceed the capacity which indicate that TFC has no consistent queueing. *In this way, **TFC can achieve zero-queueing.***

4.5 Token Adjustment

The link might be underutilized due to two main reasons. First, work-conserving problem. In topologies with multiple

bottlenecks, it is possible that a flow does not inject the traffic allocated by a bottleneck switch since another bottleneck switch allocates a smaller congestion window. Second, since the RTT of a flow includes a *random* processing delay at end hosts, the minimum measured RTT value of a flow used to compute the token value must be smaller than the average RTT value without queueing delay of the flow. According to Eq. (6), the link may be underutilized.

Therefore, to *keep high link utilization as well as avoid buffer backlog*, at the end of time slot n, the token value $T[n]$ used to compute $W[n+1]$ is adjusted according to the link utilization in time slot n.

$$T[n] = c \times rtt_b[n] \times \frac{\rho_0}{\rho[n]} \tag{7}$$

where ρ_0 is the expected link utilization and $\rho[n] = \frac{A[n]}{c \times rtt_m[n]}$ ($A[n]$ is the arrival traffic during the instananeous round trip time $rtt_m[n]$). Furthermore, to keep stable link utilization, we use the moving average method to tolerate noisy points.

$$T[n] = \alpha \times T[n-1] + (1-\alpha) \times T[n] \tag{8}$$

where α is the weight of the history token value. By using the token adjustment mechanism, not only the work-conserving problem is solved, the potential waste capacity caused by ensuring zero-queueing is compensated.

4.6 Achieving Rare Packets Loss

In data center networks, traffic bursts are likely to cause a large number of packets dropping. To achieve near zero packet loss, TFC adds a window acquisition phase and deliberately deals with the issue that the congestion window of one flow is smaller than MSS.

Traffic Bursts. Note that we do not allocate window at the establishment phase since the new arriving flows are not counted in the current congestion window computation. Assume that the SYN packet of a new flow arrives at a switch at time slot n. Then at the end of time slot n, the switch will update the number of effective flows $E[n]$ and $W[n+1]$. Thus, the flow needs another round to take back the new congestion window $W[n+1]$. Therefore, in TFC, a flow will send a marked packet without any payload after the flow establishment phase to take back proper congestion window. In this way, the packets dropping caused by new arrival highly concurrent flows can be avoided.

Window of Less Than One Packet. When the number of senders is quite large, congestion will happens even if each sender only sends one packet. Most of prior work fails to properly deal with this issue [7, 35, 38]. However, this situation is common in large-scale data centers. There is a simple solution to the problem, that is reducing MSS at the transport layer. However, the overhead caused by packets header will become quite large as the MSS decreases. One other method is that source i sends one packet every $\frac{MSS}{W_i}$ RTTs. However, to implement this solution, RTT needs to be exactly estimated and a high resolution timer to count $\frac{RTT}{W_i}$ is required. Using high resolution timer to control every packet will incur lots of interruption.

TFC solves this problem at switches enlightened by the traffic shaping mechanism, token bucket algorithm. Each switch maintains a counter for a port, which represents how many data can be sent. The counter will increase as time elapses. On a marked ACK packet arrival, if the carried congestion window is smaller than a packet and the counter value is larger than a packet, then the carried congestion window in the ACK header will be modified to one packet and the counter decreases by a packet. Otherwise, the ACK will be put into a delay queue to wait for a large enough counter. If the arrival ACK carries a congestion window that is larger than a packet, then it will be directly sent out and the counter subtracts the congestion window carried by the ACK packet. In this way, the number of flows that transmits packets during each time slot will not exceed the token value. Therefore, *near zero-queueing as well as high goodput can still be achieved.*

5. Implementation

We implemented the end host part of TFC in GNU/Linux kernel 2.6.38.3. Only the header and the congestion control mechanism are modified. The TFC header is similar to the TCP header except that it uses two reserved bits in the *flags* field to mark the first packet in each full window of packets and its ACK. The two marked bits are respectively named as RM (Round MArk) and RMA (Round MArk Acknowledgment). The switch part is implemented in NetFPGA [1].

5.1 Sender

During the establishment phase, the sender and receiver negotiate whether to use TFC or not. Then at the data transmission phase, the sender sets the RM bit in the TFC header of the first data packet to 1. After receiving a RMA marked packet, the sender sets the RM bit in the TFC header of the next sending data packet to 1. In this way, the first packet of each round has the RM indicator. TFC uses the *window* field in the packet header to carry the congestion window. Before transmitting a data packet, the sender modifies its *window* field to be *0xffff* as the initial window value.

After receiving a RMA marked packet, the sender determines its congestion window according to the window value carried in the ACK header.

5.2 Switch

The functions of switches mainly include computing tokens, measuring the number of effective flows, and updating congestion windows. Figure 3 depicts the implementation structure. A TFC switch uses 58% more logic slices (from 12807 to 20235) than the reference switch of NetFPGA project. 49.2% of the overhead is due to the use of divider. We use totally 8 dividers (each port holding two, one for window

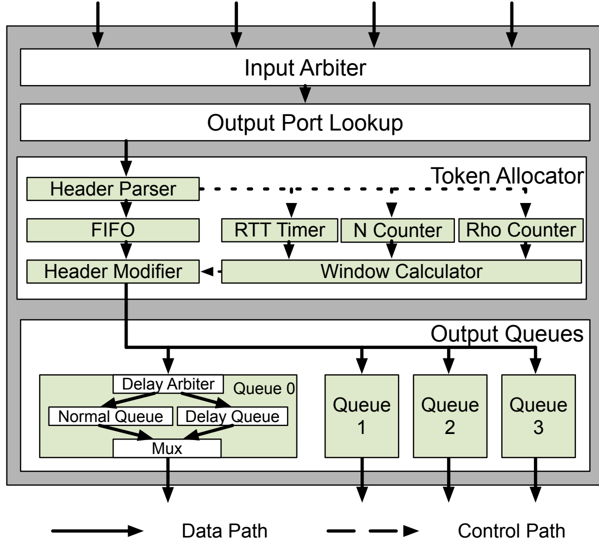


Figure 3: Implementation structure of TFC switches on NetFPGA. Packets are forwarded through the data path, and the control path forwards signals to trigger corresponding modules.

computation and the other for token adjustment). Through multiplexing, this number could then be reduced to 1. Thus, the optimized implementation of the TFC switch will introduce about 30% more logic usage compared to the reference design.

Init: Set $rtt_b = 160$ us, $E = 1$. Catch the first RM data packet, record its five tuples as the delimiter flow, and record the current time $t_{start} = t_{now}$. Set the total arrival traffic $A = 0$.

Event 1: On a data packet arrival, add the length of the data packet to the total arrived traffic A (*Rho Counter Module*). If the packet has the RM mark and does not belong to the delimiter flow, E++ (*N Counter Module*). Else if the RM data packet belongs to the delimiter flow, compute current round trip time $rtt_m = t_{now} - t_{start}$ and update $rtt_b = min\{rtt_b, rtt_m\}$ (*RTT Timer Module*), compute the link utilization during the current round trip time (*Rho Counter Module*), adjust the token value according to eq. (7) and compute the congestion window $W = \frac{T}{E}$ (*Window Calculator Module*). Let $E = 1$ and $t_{start} = t_{now}$.

Event 2: On an ACK packet with a RMA mark arrival, if the carried congestion window is less than MSS, it will be delayed according to the algorithm described in Section 4.6 (*Delay Arbiter Module*).

When the current delimiter flow ends. If the current delimiter flow ends, then switches will never catch a RM marked packet with the same five tuples of the current delimiter flow and thus the congestion window will never be updated. In TFC, the FIN packet and a timer are used to solve this problem. If a FIN packet is received or $2^k \times rtt_{last}$ has

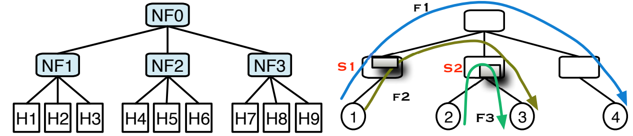


Figure 4: Testbed.

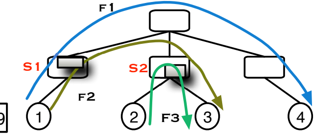


Figure 5: Work-conserving scenario.

past (k is the miss times) and no boundary RM flow comes, TFC will catch another proper RM packet as the new delimiter flow. Thus, at the first time, a new RM will be catched after $2 \times rtt_{last}$; at the second time, a new RM packet will be catched after $4 \times rtt_{last}$, and so on. The maximum k is set to 7. The function is implemented in the *RTT Timer Module*.

5.3 Receiver

After receiving a data packet with a RM mark, the minimum of the advertised congestion window size, *awnd*, at the receiver and the *window* value carried in the header of the RM data packet is assigned to the *window* field of the corresponding ACK header. Besides, the RMA bit in the ACK header is set to 1.

6. Evaluation

6.1 Small-Scale Experiment

6.1.1 Experimental Setup

The performance of TFC is evaluated in a small testbed as shown in Figure 4. The testbed is consisted of 4 NetFPGA boards and 9 servers. Each server is a DELL OptiPlex 360 desktop 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 board hosting in a DELL server has a buffer of 256 KB per port and four 1 Gbps ports.

The performance of TFC is evaluated in different scenarios and is compared with TCP NewReno and DCTCP. The threshold of marking packets, K, is set to 32 KB in DCTCP and the weighted averaging factor, g, is set to $1/16$ as recommended in [7]. The parameter of TFC, ρ_0, is set to 0.97, α is set to $\frac{7}{8}$.

The performance evaluation mainly includes four parts. First, the mechanisms of measuring the token value and N_e are evaluated to check whether the measured results are accurate. Second, the basic performance of TFC is evaluated using a series of micro-benchmarks. Third, we evaluate TFC using the benchmark traffic generated from the measured data in [7]. Finally, large-scale ns-2 simulations are conducted to validate that TFC can work well in large-scale topology.

6.1.2 Experimental Results

Accuracy of Measuring rtt_b and N_e. First, a series of experiments are conducted to validate that TFC is able to obtain relatively accurate rtt_b and N_e.

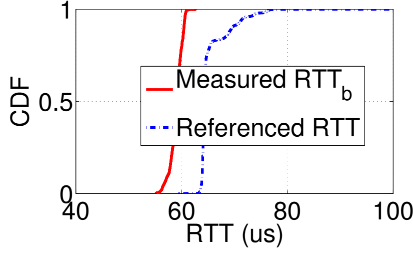


Figure 6: Measured rtt_b.

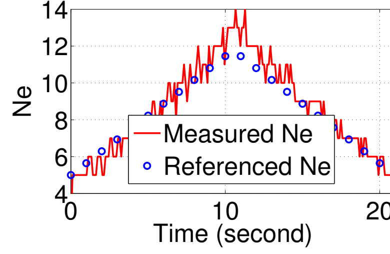


Figure 7: Accuracy of N_e with inactive flows.

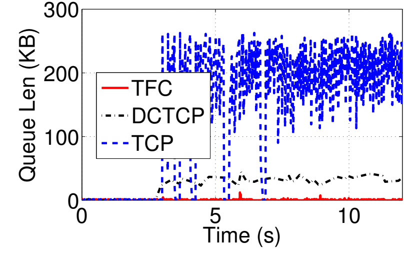
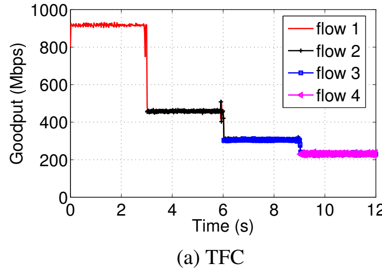
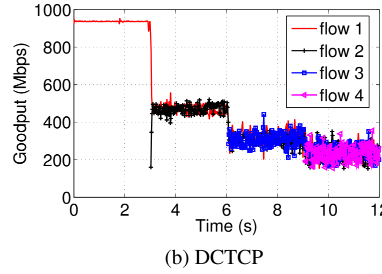


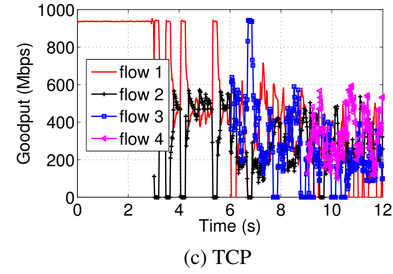
Figure 8: Queue length.



(a) TFC

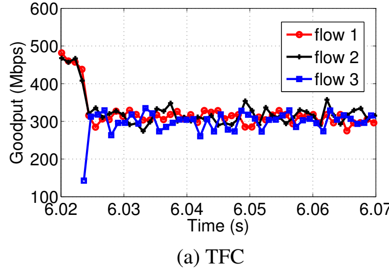


(b) DCTCP

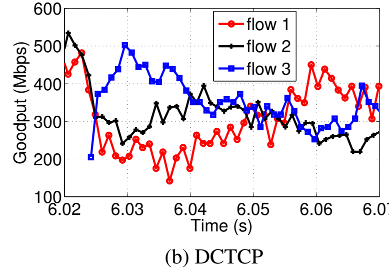


(c) TCP

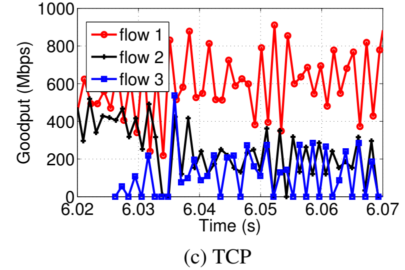
Figure 9: High Goodput and Fairness.



(a) TFC



(b) DCTCP

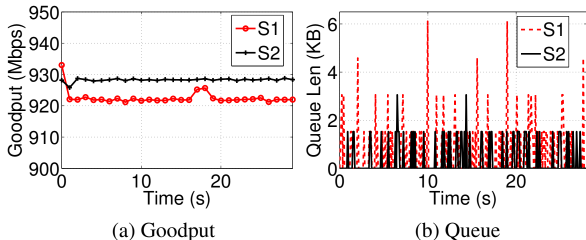


(c) TCP

Figure 10: Convergence rate.

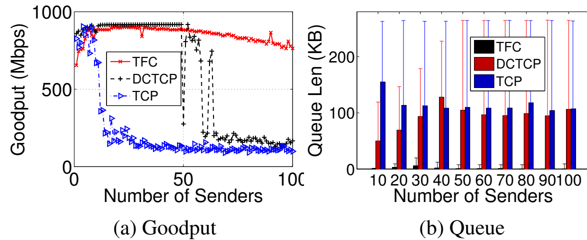
First, hosts H_1 and H_2 send 2 long-lived flows to host H_3, respectively. rtt_b is measured at the interval of 1 second, that is, rtt_b is set to the minimum of the measured rtt_m during 1 second. Besides, we let host H_1 send one packet with MTU of 1500 Bytes to H_3 per round trip time and get referenced rtt values. Figure 6 shows the CDF of the measured rtt_b and the referenced rtt. It shows that the measured rtt_b is around 59 microseconds, while the referenced rtt is about 65 microseconds. This is because the round trip time of a flow varies due to random processing time at end hosts. The measured rtt_b does not include the random processing delay. However, we can see that the difference between rtt_b and the referenced rtt is relatively constant. Thus, we could use the token adjustment mechanism to get the precise value of tokens.

Then we let hosts H_1 and H_4 set up n_1 and n_2 flows to host H_6. Switch NF2 measures the number of effective flows at the port connecting to host H_6. The delimiter flow of counting N_e is a flow sent from host H_4. To investigate whether the method can exclude inactive flows, we let $n_2 = 5$, and n_1 gradually increase from 1 to 10 and then gradually become inactive at the interval of 1 second. Figure 7 depicts the measured number of effective flows. The measured value is sampled every 0.1 second. The round trip time of flows from host H_1 is about 1.5 times of the delimiter flow from host H_4 according to our experimental data. Thus, the expected number of effective flows is $\frac{n_1}{1.5} + n_2$ according to eq. (1). The plotted results show that the measured N_e is quite close to the computed value. Also, the variance of the samples is small.



(a) Goodput (b) Queue

Figure 11: Work-conserving.



(a) Goodput (b) Queue

Figure 12: Incast: goodput and queue length.

High Goodput. Let hosts H_1 and H_2 establish 2 flows to host H_3 at the interval of 3 seconds, respectively. Figure 9 draws the goodput curves of different flows. The goodput values are sampled every 20 milliseconds. The results indicate that all of TFC, DCTCP and TCP are able to fully utilize the bottleneck bandwidth. In terms of fairness among flows, TFC flows fairly share the whole bandwidth even if in small timescale, while the goodput of TCP flows is quite unstable. DCTCP performs much better than TCP since it avoids large numbers of packets loss by limiting the queue length.

Zero Queueing. Figure 8 shows the queue length variation with different mechanisms in the above scenario. At the first 3 seconds, all the three mechanisms have zero queue length since one flow does not accumulate data packets. Afterwards, TFC indeed achieves near zero queueing delay. Several instantaneous queue length is a little large, but the maximum queue length is only about 9 KBytes. DCTCP limits the queue length at about 30 KBytes. TCP flows fill up the whole buffer and the queue length is about 256 KB.

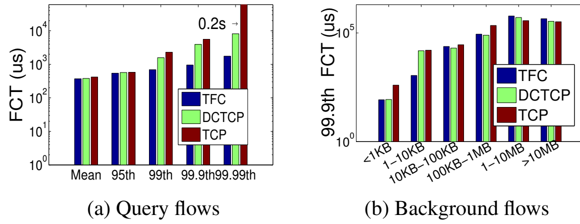
Fast Convergence. To investigate the convergence rate in detail, we amplify the goodput curves from the start of flow 3 in Figure 10. TFC flows converge to the expected goodput quite quickly since they can get their fair share of bandwidth in one round. In DCTCP, flow 3 spends about 20 microseconds to converge to the fair share. While most of short flows in data centers can be finished in 10 milliseconds [7]. TCP flows hardly converge to the fair share in a short time.

Work-Conserving. To validate that TFC does not have the work-conserving problem, we conducted experiments in the topology as shown in Figure 5. we let host 1 initiate n_1 flows to host 4 and n_2 flows to host 3. Host 2 sends n_3 flows to host 3. Therefore, two bottleneck links form. One is the uplink connecting switch S1, the other one is the downlink connecting S2. Let $n_1 = 8$ and $n_2 = n_3 = 2$. The goodput of each flow is sampled and the queue length variation at the bottleneck ports is measured. Since switch $S2$ allocates higher congestion window to n_2 flows than $S1$ does, the link of S2 could not be fully utilized if the work-conserving problem happens. Figure 11 shows the aggregated goodput and queue length variation at switches S1 and S2. Both S1 and S2 achieve high goodput, which indicates that *TFC does not suffer the work-conserving problem.* Note that the goodput of S1 is a little smaller than S2. This is because each flow has the same congestion window and flows n_3 pass fewer hops than flows n_2. Thus, the goodput of flows n_3 is a little larger than flows n_2. Figure 11b shows the queue length variation at S1 and S2. We can see that the queue length varies around 2 KB, which is about one packet. *TFC achieves near-zero queueing.*

Bursty Fan-in traffic. The incast communication pattern likely causes throughput deterioration due to the fan-in bursts. We generate the Incast traffic according to the description in previous work [36]. A receiver requests data blocks to a certain number of senders. The senders synchronously respond data blocks to the receiver. The receiver could not request the next round data blocks until it receives all the current transmitted data blocks. In our experiment, the block size is 256 KB. The receiver requests data blocks to all the senders for 1000 times. Figure 12a depicts the goodput with different number of senders in the incast communicaiton pattern. Note that a server possibly acts as several senders in the experiment since the number of physical servers is limited in our testbed. We can see that the goodput of TFC with different number of senders is about 800-900 Mbps. DCTCP achieves high goodput when the number of senders is smaller than 50 and suffers goodput collapse as the number of senders is more than 50. TCP exhibits the worst performance. Its goodput decreases dramatically as the number of senders is larger than 10.

Figure 12b draws the average and maximum queue length vs. the number of senders. *TFC almost has no buffer backlog.* The maximum queue length of TCP is close to the buffer length per port, 256 KB, since the window decrease of TCP is loss-driven. The average and maximum queue length of DCTCP is smaller than 100 KB and 200 KB when the number of senders is smaller than 40, respectively. However, as the senders increase, the queue length of DCTCP is similar to TCP.

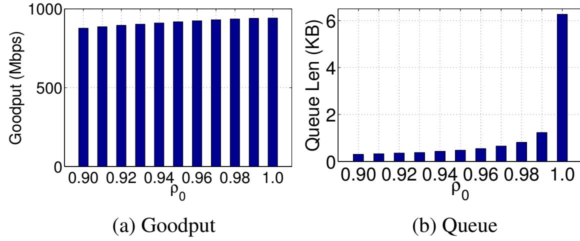
Benchmark. To evaluate the performance of TFC with more realistic workload. We conducted experiments in the small testbed shown in Figure 4. First, we generated realistic traffic, including query, short messages and background flows, based on the cumulative distribution function of the interval time between two arrival flows and the probability distribution of background flow sizes in [7]. The distribution



(a) Query flows

(b) Background flows

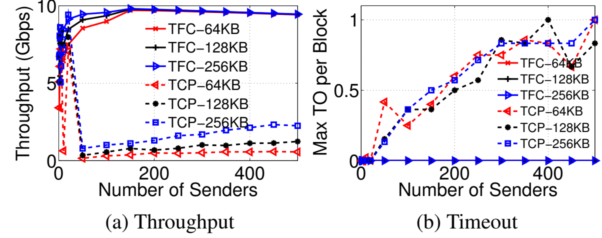
Figure 13: FCT of flows.



(a) Goodput

(b) Queue

Figure 14: Impact of ρ.



(a) Throughput

(b) Timeout

Figure 15: Incast communication pattern.

curves are gotten based on a large amount of measured data from 6000 servers in a real data center network [7]. The size of each query message is 2 KB. Then servers in the small testbed transmit data according to the time interval and flow size in the generated traffic.

Figure 13a depicts the flow completion time of query flows. The average and tail flow completion time under T-FC is much smaller than DCTCP and TCP. Especially, the 99.99th percentile flow completion time of TCP is quite large since some flows suffer timeouts. Figure 13b shows the flow completion time of background flows. TFC flows that are smaller than 10KB finish quickly than DCTCP and TCP. While other flows finish a little slower than DCTCP. This is because the link capacity is constant. Query flows take more bandwidth. Thus some background flows use less bandwidth.

Parameters. Hosts H_1-H_5 establishes a flow to host H_6, respectively. Let ρ_0 vary from 0.9 to 1.0. Figure 14 depicts the goodput at the receiver and the queue length at the port connecting to host H_6. We can see that the goodput at the receiver matches the expected link utilization, ρ_0. As ρ_0 increases from 0.90 to 1.0, the goodput of the receiver increases from 880 Mbps to 940 Mbps. The packet header takes the additional bandwidth. Figure 14b shows that only less than 1 KBytes queue length exists when $\rho_0 < 0.98$. This small queue length is unavoidable due to the store and forward process at switches. When ρ_0 is larger than 0.98, the queue length gets larger. Especially when $\rho_0 = 1.0$, the average queue length becomes about 6 KB. This is because the instantaneous round trip time rtt_m varies, if the expected link utilization is 100%, possibly some packets will be accumulated in the buffer if the current round trip time is smaller than the average round trip time.

6.2 Large-Scale Simulation

To evaluate the performance of TFC in large-scale networks, TFC is implemented on the ns-2 platform and simulations are conducted with higher bandwidth and larger number of flows.

6.2.1 Bursty Fan-in Traffic

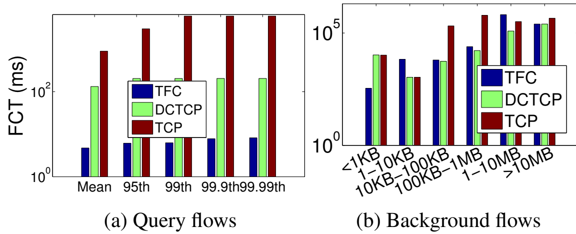
We generate the Incast communication pattern in large-scale simulation platform to emulate bursty fan-in traffic. Each link has the rate of 10 Gbps. The switch buffer size is 512 KB. The synchronized block size is 256 KB. At start, the receiver sends a request to all the senders, then each sender transmits 256 KB data to the receiver. After successfully receiving the blocks from all the senders, the receiver will send requests to all the senders for the next round of blocks. The simulation lasts for 2 seconds, and we analyzed the averaged goodput, the total number of timeouts, and the instantaneous queue length.

Figure 15a depicts the averaged throughput at the receiver with different block size. The link utilization of TFC is always around 90% with different number of senders, while the throughput of TCP dramatically decreases with more than 50 senders. Note that the link utilization in both TFC and TCP is a little low when the number of senders is quite small. This is because the time used to transmit blocks is quite small. For example, if there is one sender and the data block size is 256 KB, then the data only needs two rounds to finish. However, the request sent by the receiver to notify all the senders transmit data will waste a round. Thus, the link can not be used to transmit blocks all the time.

Figure 15b depicts the maximum timeouts suffered by one flow per block. In TFC, *the number of timeouts is always around zero no matter how many senders concurrently transmit data blocks since TFC could achieve near-zero packet loss*. This explains why TFC achieves high throughput with different number of senders. While TCP flows suffer lots of timeouts. When the number of senders is larger than 300, one block will suffer 0.8 timeout on average.

6.2.2 Benchmark

The topology is similar to the real testbed as shown in Figure 4 except that the number of leaf switches increases to 18 from 3. And each leaf switch is connected to 20 servers.



(a) Query flows (b) Background flows

Figure 16: Simulation: FCT of flows.

Each leaf switch and its connected servers constitutes a rack. Each leaf switch has 20 1Gbps downlinks to the servers and 1 10Gbps uplink to the top switch. The latency of each link is 20 us. Thus, the end-to-end round trip latency of 4 hops (inter-rack flows) is 160 us and the end-to-end round trip latency of 2 hops (intra-rack flows) is 80 us.

The benchmark workload is generated using the same method as that in the experiment. Figure 16a shows the flow completion time of different kinds of flows. On average, the flow completion time of one DCTCP query flow is about 30 times more than one TFC flow, and one TCP query flow uses about 8 times more time to finish than one DCTCP query flow. Besides, the tail latency of TFC is quite small. However, DCTCP and TCP suffer much high tail latency. This is because there are 360 servers in our simulation. One query request will cause 359 servers transmit a query response to the last server concurrently. *This 359 concurrently flows likely overwhelm the bottleneck buffer under DCTCP and TCP. However, TFC can effectively deal with this high traffic burst due to its delay function at switches.*

Figure 16b depicts the flow completion time of background flows. When the flow size is larger than 1 KB, TFC performs a little worse. This is because TFC query flows do not suffer timeouts and thus they can take more bandwidth compared with DCTCP and TCP. While in DCTCP and TCP, a large number of timeouts suffered by query flows cause that query data could not fill network links all the time. Therefore, background flows can take more bandwidth.

7. Related Work

Data Center Transport Control. DCTCP[7] and D^{2}TCP [35] follow the framework of TCP and leverage the ECN marking scheme to keep the queue length low. However, they are not responsive for short flows due to slow convergence.

D^3 [37] allocate bandwidth to flows according to their deadline demand and distribute the rest fairly to every flow. However, D^3 is a clean slate design which not only need specialized switch, but also need to modify the application. Besides, D^3 will lead to cumulative error in counting the number of flows.

Fastpass [31] uses a centralized arbiter to determine the time at which each packet should be transmitted as well as the path to use for that packet. However, the scalability of Fastpass is limited by the centralized arbiter that needs to deal with all the packets.

There are several work use flow scheduling to optimize flow completion time in data center [9, 11, 20, 21, 30]. These work focus on the benefits of scheduling and use legacy protocol or simple congestion control to simplify their design. While, TFC is a new which is orthometric to them.

HULL achieves near zero buffer through phantom queue, while it based on DCTCP which cannot provide extreme fast convergence and rare packet loss.

Credit-Based Flow Control. Kung *et al.* proposed a series of work on credit-based flow control mechanisms for ATM networks to to achieve near zero loss ratio [24–28]. The credit-based flow control mechanism works hop-by hop. Each receiver notifies its sender to transmit a certain amount of data cells by sending credit cells. After having received a credit cell, the sender can forward data cells according to the received credit value. In this way, the credit-based flow control achieves near zero loss ratio. However, the credit-based flow control mechanism requires to maintain a separate buffer space for each session (VC) passing through the link. The number of connections sharing a path is usually large in data centers. Maintaining the buffer space for all connections will introduce high overhead.

Explict Flow Control. There are some typical explicit transport protocols designed for traditional Internet [17, 23]. However, they are not suitable for data center networks. For example, XCP [23] is designed for networks with high bandwidth-delay product, the convergence rate of it is quite slow with respect to the fast convergence requirement in data center networks [17]. RCP [17] has large buffer requirement during flow join and hardly deals with large-scale concurrent flows [40]. Besides, the convergence rate of it is also slow [40].

8. Conclusion

In this work, an explicit window-based transport protocol, called TFC, is proposed for data center networks to achieve high link utilization, fast convergence, zero-queueing and rare packets loss. Two concepts, Token and Effective Flows, are defined to represent the network resource to be allocated and resource consumers in a time slot. By excluding in-network buffer from tokens and decoupling the measuring mechanisms for determining tokens and the number of effective flows in a time slot, TFC achieves near zero-queueing. Besides, by adding a window acquisition phase after the flow establishment phase and a delay function at switches, TFC prevents packets dropping with traffic bursts and highly concurrent flows and thus achieves rare packets loss. The experimental results in our testbed and simulation results indicate that TFC achieves its goals.

Acknowledgments

We gratefully appreciate our shepherd Prof. Jon Crowcroft for his constructive suggestions, and acknowledge the anonymous reviewers for their valuable comments. This work is supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61225011 and No. 61502049, National Basic Research Program of China (973 Program) under Grant No.2012CB315803, and National High-Tech Research and Development Plan of China (863 Plan) under Grant No. 2015AA020101.

References

[1] NetFPGA project. http://netfpga.org/.

[2] Yahoo! M45 supercomputing project. http://research.yahoo.com/node/1884, 2009.

[3] Google Research: Three things that MUST BE DONE to save the data center of the future. http://www.theregister.co.uk/2014/02/11/google_research_three_things_that_must_be_done_to_save_the_data_center_of_the_future/?page=3, Feb. 2014.

[4] Storm: Distributed and fault-tolerant realtime computation. http://storm-project.net/, January 2014.

[5] D. Abts and B. Felderman. A Guided Tour through Datacenter Networking. *ACM Queue*, 10(5):10, 2012.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM*, 2008.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM*, 2010.

[8] 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 *Proceedings of the USENIX NSDI*, 2012.

[9] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal Near-Optimal Datacenter Transport. In *Proceedings of the ACM SIGCOMM*, 2013.

[10] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-Scale Key-Value Store. In *Proceedings of the ACM SIGMETRICS*, 2012.

[11] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *Proceedings of the USENIX NSDI*, 2015.

[12] L. A. Barroso, J. Dean, and U. Holzle. Web Search for a Planet: The Google Cluster Architecture. *Micro*, 23(2):22–28, 2003.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *Proceedings of the ACM CoNEXT*, 2011.

[14] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009.

[15] P. Cheng, F. Ren, R. Shu, and C. Lin. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. In *Proceedings of the USENIX NSDI*, 2014.

[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the USENIX OSDI*, 2004.

[17] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor Sharing Flows in the Internet. In *Quality of Service–IWQoS 2005*. 2005.

[18] B. Fitzpatrick. Distributed Caching with Memcached. *Linux journal*, 2004(124):5, 2004.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM*, 2009.

[20] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues Dont Matter When You Can JUMP Them! In *Proceedings of the USENIX NSDI*, 2015.

[21] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *Proceedings of the ACM SIGCOMM*, 2012.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009.

[23] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-delay Product Networks. *Proceedings of the ACM SIGCOMM*, 2002.

[24] H. Kung and K. Chang. Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks. In *IEEE INFOCOM*, pages 239–252, 1995.

[25] H. Kung and A. Chapman. The FCVC (Flow-Controlled Virtual Channels) Proposal for ATM Networks: A Summary. In *IEEE ICNP*, pages 116–127, 1993.

[26] H. Kung and R. Morris. Credit-Based Flow Control for ATM Networks. *IEEE Network*, 9(2):40–48, 1995.

[27] H. Kung and S. Y. Wang. Client-Server Performance on Flow-Controlled ATM Networks: a Web Database of Simulation Results. In *IEEE INFOCOM*, pages 1218–1226, 1997.

[28] H. Kung and S. Y. Wang. Zero Queueing Flow Control and Applications. In *IEEE INFOCOM*, pages 192–200, 1998.

[29] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav. Quartz: a New Design Element for Low-Latency DCNs. In *Proceedings of the ACM SIGCOMM*, 2014.

[30] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not Foes: Synthesizing Existing Transport Strategies for Data Center Networks. In *Proceedings of the ACM SIGCOMM*, 2014.

[31] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized Zero-Queue Datacenter Network. In *Proceedings of the ACM SIGCOMM*, 2014.

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

[33] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the IEEE INFOCOM*, 1999.

[34] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, et al. Storm@ Twitter. In *Proceedings of the ACM SIGMOD*, 2014.

[35] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-Aware Datacenter TCP (D2TCP). In *Proceedings of the ACM SIGCOMM*, 2012.

[36] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *Proceedings of the ACM SIGCOMM*, 2009.

[37] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better Never than Late Meeting Deadlines in Datacenter Networks. In *Proceedings of the ACM SIGCOMM*, 2011.

[38] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *Proceedings of the 6th ACM CoNEXT*, 2010.

[39] J. Zhang, F. Ren, and C. Lin. Modeling and Understanding TCP Incast in Data Center Networks. In *Proceedings of the IEEE INFOCOM*, 2011.

[40] Y. Zhang, S. Jain, and D. Loguinov. Towards Experimental Evaluation of Explicit Congestion Control. *Computer Networks*, 53(7):1027–1039, 2009.