Computer Communications 69 (2015) 60–68

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Exploring parallelism and desynchronization of TCP over high speed networks with tiny buffers

Cheng Cui, Lin Xue, Chui-Hui Chiu, Praveenkumar Kondikoppa, Seung-Jong Park*

School of Electrical Engineering and Computer Science, Center for Computation and Technology, Louisiana State University, USA

ARTICLE INFO

Article history:
Available online 17 July 2015

Keywords:
TCP congestion control
High speed networks
Tiny buffer at routers
TCP synchronization

ABSTRACT

The buffer sizing problem is a big challenge for high speed network routers to reduce buffer cost without throughput loss. The past few years have witnessed debate on how to improve link utilization of high speed networks where the router buffer size is idealized into dozens of packets. Theoretically, the buffer size can be shrunk by more than 100 times. Under this scenario, widely argued proposals for TCP traffic to achieve acceptable link capacities mandate three necessary conditions: over-provisioned core link bandwidth, non-bursty flows, and tens of thousands of asynchronous flows. However, in high speed networks where these conditions are insufficient, TCP traffic suffers severely from routers with tiny buffers.

To explore better performance, we propose a new congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP) that creates a flow with parallel channels. These channels can desynchronize each other to avoid TCP loss synchronization, and they can avoid traffic penalties from burst losses. Over a 10 Gb/s large delay network ruled by tiny buffer routers, our emulation results show that bottleneck link utilization can reach over 80% with much fewer number of flows. Compared with other TCP congestion control variants, DMCTCP can also achieve much better performance in high loss rate networks. Facing the buffer sizing challenge, our study is a new step towards the deployment of optical packet switching networks.

1. Introduction

Router buffer needs to be large enough to accommodate the dynamics of TCP congestion control. The traditional guidance on maintaining a fully utilized link while TCP ramps up its congestion window suggests a Bandwidth-Delay-Product (BDP). Equivalently, this *rule-of-thumb* decides the amount of buffering by $B = C \times \text{RTT}$ [1], where C is the capacity of a bottleneck link and RTT is the Round-Trip-Time of a TCP connection flowing through the router. However, for a typical RTT of 250 ms, a router with a $C = 40\,\text{Gb/s}$ link capacity requires 10 Gb of buffering, which poses considerable challenges to design and cost of networks.

As BDP keeps growing, larger and more expensive buffers will also exert higher interference on TCP flows. Studies [2,3] argue that large buffers tend to induce TCP loss synchronization, because large buffers prolong TCP's control loop and enlarge queuing dynamics. Recent studies [4,5] proposed a *tiny-buffer model* to significantly reduce buffer to a size of $O(\log W)$, where W is congestion window size. They recommended that a few dozen packets of buffering can suffice an acceptable link load for TCP traffic (e.g., 75% utilization). Theoretically, the buffer size can be shrunk by more than 100 times. This model has been examined with promising results from several 1 Gb/s network experiments [6].

Although the above works suggest that a tiny buffer may be sufficient for a relatively large bandwidth, they assume that tens of thousands of TCP flows are neither bursty nor synchronous. In fact, such traffic relies on the Internet backbones, where tens of thousands of flows are spread out through over-provisioned core links. In addition to the backbone networks, nowadays most universities and research laboratories provide high speed access networks to support large scale scientific research. The bandwidth for such networks has been growing increasingly in the order of Gb/s. In a typical setup, routers have only a very small amount of buffers, but it is still not clear whether such small amount of buffers is sufficient to serve the fast growing bandwidth.

We argue that existing tiny buffers are not sufficient to meet the bandwidth requirement in high speed networks especially in the access networks. This is because: (i) the access links have congestion, (ii) the network traffic is bursty, and (iii) the number of flows is at least one order of magnitude smaller to reach an ideal level of asynchronism.

Therefore, it is critical to provide a TCP solution for high speed access networks with tiny buffers. First, the solution can meet the

* Corresponding author. Tel.: +1 225 578 2209; fax: +1 225 578 1465.
E-mail addresses: ccui@cct.lsu.edu (C. Cui), xuelin@cct.lsu.edu (L. Xue), cchiu1@lsu.edu (C.-H. Chiu), pkondi1@lsu.edu (P. Kondikoppa), sjpark@cct.lsu.edu, trysjp@hotmail.com (S.-J. Park).

http://dx.doi.org/10.1016/j.comcom.2015.07.010
0140-3664/© 2015 Elsevier B.V. All rights reserved.

bandwidth requirement of end users that have high speed connectivity. Second, it can reduce the challenge of deploying all-optical routers that are limited by buffer size [7] but succeeding in huge bandwidth and low power cost. Third, it can reduce router complexity, making them easier to build and easier to scale. And last but not least, it can minimize the queuing delay and jitter that are intrinsic to buffer size.

We propose a new TCP congestion control algorithm called Desynchronized Multi-Channel TCP (DMCTCP). DMCTCP creates a flow with multiple channels. It desynchronizes channels and recognizes burst congestion. Therefore, impacts of TCP loss synchronization and burst losses are avoided. The key ideas behind it are to prevent simultaneous sending rate cuts from loss synchronization and to prevent sending rate penalties from burst losses. The algorithm is inspired by parallel TCP [8] and Multi-Path TCP [9] (MPTCP), but with the important distinctions that various congestion events are detected and distinguished with different corresponding actions, and that no modifications to or assistance from other layers (e.g., the link layer) are required.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we first present typical TCP loss synchronization based on an idealized model of TCP. Then, we derive a loss desynchronization solution and illustrate performance improvement. Second, we propose two types of congestion, and suggest that they should be differentiated in TCP congestion control. In Section 4, we specify the design of DMCTCP that yields robust performance. We also show the algorithm implementation that takes advantage of the MPTCP release supported in the Linux kernel. Compared DMCTCP with other TCP variants, Section 5 demonstrates our 10 Gb/s experimental results. The last section concludes the paper.

2. Background and related work

TCP loss synchronization has been studied for many years. It is firstly defined in [10] as that when one flow of a pair has congestions, the synchronization events are these congestions shared with the second flow in the same RTT. Studies [11,12] confirmed the existence of TCP loss synchronization and gave quantitative evaluation through Internet experiments and software simulation.

Traffic burstiness can be exaggerated by various packet offload mechanisms that reduce CPU overhead. For example, Interrupt-Coalescing and TCP-Segmentation-Offloading are standard features that save CPU cycles and allow the network interface card (NIC) to do the job of segmentation. Studies [6,13] illustrated that they were detrimental to network performance. Later studies proposed expensive solutions such as disabling the above mechanisms, specializing hardware to pace flows [14,15], or enabling Data-Center TCP (DCTCP) [16] over Explicit-Congestion-Notification (ECN). However, paced flows sometimes suffer in performance when competing with non-paced short flows [17], and DCTCP requires explicit feedback from middleboxes that are not widely deployed.

Active-Queue-Management (AQM) has been an active area of research over TCP desynchronization. However, when the router buffer size is reduced to a few dozen of packets, AQMs (such as Random-Early-Detection (RED) [18]) are not reliable to desynchronize flows when the buffer is small and the line rate is large [2,7,13,19].

Queuing delay also becomes negligible in such tiny router buffers. It is clear that only loss based TCP congestion control variants work well in tiny buffer networks because pure delay based and loss-delay based (hybrid) TCP variants [20–23] require detectable queuing delay as the full or partial symptom of congestion. Among loss-based TCP variants, TCP-SACK and TCP-CUBIC [24] are widely deployed as standard TCP algorithm and default TCP of Linux respectively. These early algorithms do not consider TCP desynchronization.

The concept of parallel TCP enables applications that require good network performance to use parallel TCP streams. PSockets [8] at application level opens concurrent TCP streams to increase aggregate TCP throughput. However, parallel TCP streams are not only aggressive [25], but they are also not aware of TCP loss synchronization and burst congestion. Multi-Path TCP (MPTCP) [9] is a proposed TCP extension to use multiple linked paths for a single TCP connection. It manages data streams called subflows among multiple paths. In Linux kernel, it currently has two implemented congestion control algorithms named MPTCP-Coupled [26] and MPTCP-OLIA [27]. MPTCP is likely to work correctly in the Internet through different middleboxes [28]. Study in [29] shows robust performance of MPTCP with a few subflows per-flow in a multiple fat-tree-topology data center environment. But the related buffer sizing issues are not discussed.

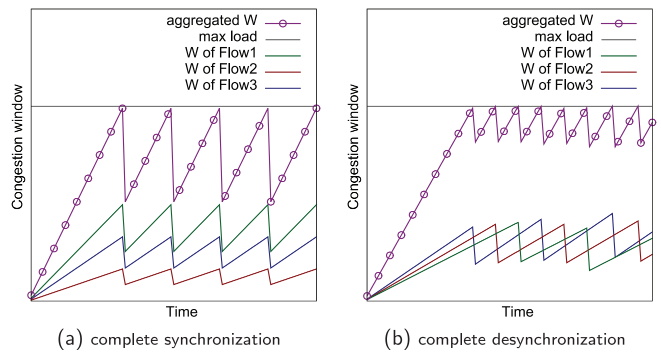

Fig. 1. Heuristic comparison of the aggregated traffic throughput between TCP synchronization and TCP desynchronization.

3. Motivation: problems of high speed access networks with tiny buffers

3.1. The TCP loss synchronization analysis

From the standard Additive-Increase/Multiplicative-Decrease (AIMD) TCP congestion control, a simplified macroscopic model for the steady-state [30,31] is expressed as follows:

$$\text{average throughput of a TCP flow} = \frac{0.75 \cdot W}{RTT} \tag{1}$$

Consider a particular RTT and segment size, the average throughput of a flow will be roughly 75% of its largest congestion window W. If buffer depth is negligible, W will be cut in half when the sending rate reaches the bottleneck link capacity C. Then it increases by one segment per-RTT until again reaches W. Therefore, the average throughput of a flow in a bufferless bottleneck link is simplified as:

$$\text{average throughput of a TCP flow} = 0.75 \cdot C \tag{2}$$

Let a complete TCP loss synchronization event happen when all the flows experience packet drops in a congestion event. As a result, all the flows cut their rates in half and the bottleneck link is underutilized at the time. This congestion event includes at least n packet drops where n is the number of flows. When n is small, it is highly probable to have many complete loss synchronization events. Fig. 1a geometrically shows a generalized scenario of complete loss synchronization. As can be seen from the figure, the aggregated traffic window of three flows is cut in half on each congestion event and it follows the same sawtooth pattern, no matter what RTT each flow has. Therefore, the bottleneck link utilization becomes 75%.

The above analyses trigger our goal to desynchronize TCP flows. In order to desynchronize, each congestion event should ideally include one packet drop such that only one flow takes a rate cut. Fig. 1b shows this idealized TCP desynchronization that improves traffic throughput. However, it is impractical to drop only one packet when the buffer overflows, especially for a tiny buffer that holds only a few dozen packets. This means Active-Queue-Management (AQM) mechanisms such as Random-Early-Detection (RED) will not work efficiently. They will behave mostly like a drop-tail buffer because when the buffer is too small, they cannot absorb the large bursts due to the faster window growth inherent in TCP protocols [2,13,19]. Therefore, instead of AQM, TCP congestion control becomes the target to find a desynchronization solution.

As shown in Fig. 1b, dropping the flow that has the largest congestion window will balance fairness among multiple flows [32]. This requires communication among flows so that smaller flows should not cut rate at the time when the largest flow cuts off. However, none of the early TCP variants can satisfy this requirement as these variants manage single-flow per-connection.

3.2. The burstiness analysis

In high speed access networks with tiny buffers, the link utilization is most likely to be lower under many contending TCP flows. Not only because of TCP loss synchronization, but also because of the inherent flow burstiness. Worse still, by applying techniques to save CPU overhead, such as Interrupt-Coalescing and TCP-Segmentation-Offloading, flow burstiness is exaggerated. As a result, the normal TCP ACK-clocking is disrupted and packets are burst out of the NIC at line rate [6,13]. Because these techniques have become desirable for high speed networks beyond 10 Gb/s, burstiness is not avoidable. Also, it induces complex and expensive solutions to pace packets.

We consider two types of congestion: (a) bandwidth congestion, which is caused by the high utilization of bottleneck link among competing flows and (b) burst congestion, which is caused by random burst contention that occurs even when bandwidth utilization is low. We believe the impact of the second type should be avoided if such congestion can be distinguished. This brings another challenge because most loss-based TCP variants use packet losses (duplicated ACKs) as a signal of bandwidth congestion and verify burst congestion is hard work [33].

4. The DMCTCP congestion control

Inspired by parallel TCP and MPTCP, DMCTCP pursues minimal TCP loss synchronization and reduces impact of burst congestion. Similar to a MPTCP flow's subflows used among available paths, a DMCTCP flow consists of multiple channels. But these channels carry split data from upper layer through *a single path* and reassemble the data at the receiver. Each channel has an individual congestion window. By comparing channels through communication, the algorithm can detect and distinguish loss synchronization and burst congestion.

4.1. DMCTCP design in detail

Let m $(m \geq 1)$ be the number of channels of a DMCTCP flow through a single path. Obviously, at least two channels are required to compare with each other when congestion happens. We denote by w_i the congestion window of channel i $(i \in [1, \ldots, m])$; by $w_{max} = \max\{w_i, i \in [1, \ldots, m]\}$, $w_{min} = \min\{w_i, i \in [1, \ldots, m]\}$ and $w_{total} = \sum_{i=1}^{m} w_i$ the largest, the smallest and the aggregated congestion window of all the channels at time t; by $time_i$ the time stamp of a detected loss in channel i; and by $time_c$ the time stamp of a most recent rate cut of any channel. We assume all channels have the same Round-Trip-Time (rtt) because they are through the same path with a negligible buffer depth. The algorithm is as follows:

- Establish only one channel that follows standard slow-start when a connection begins.
- Add $(m-1)$ channels in steady-state and
  - for each loss on channel $i \in m$, decrease w_i by:
    - $w_i/2$ when $((w_i = w_{max})$ && $(time_i - time_c) > rtt)$, then $time_c \leftarrow time_i$;
    - 0, otherwise.
  - for each ACK on channel $i \in m$, increase w_i by:
    - $1/w_i$ when $w_i = w_{min}$;
    - $1/(w_{total} - w_{min})$, otherwise.

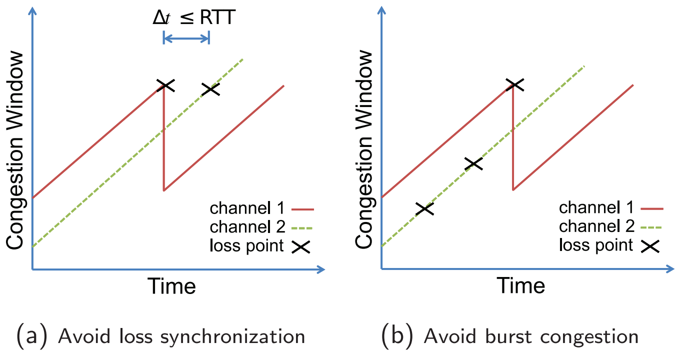

Fig. 2. Congestion control scenarios of a DMCTCP flow with two channels.

DMCTCP is a loss-based algorithm. At the beginning, only one channel is used in slow-start phase such that it is compatible with a short TCP flow. In stable phase on the decrease part, as illustrated in Fig. 2a, the algorithm decides only the largest channel can be cut in half in a congestion event, and it desynchronizes channels by guaranteeing consecutive cuts are avoided within the same congestion event (one rtt interval). As explained in studies [10–12], the granularity of one RTT is used to determine if two losses are in the same congestion event. As illustrated in Fig. 2b, when the largest channel has no loss in a congestion event, the algorithm determines all the losses of the smaller channel in the same congestion event are caused by burst congestion. This is because the largest channel will be more likely to encounter the bandwidth congestion than the smaller channels; and it will be unfair for the smaller channels to yield their bandwidth shares when the largest channel does not. Therefore, the unnecessary rate cuts on smaller channels are avoided. In other words, the largest channel can be used to probe bandwidth congestion while the other smaller channels can be used to detect burst congestion. On the increase part, for simplicity and compatibility, we choose the same increase rate for the case of w_{min} channel and the case of rest channels. This guarantees that when only one channel per-flow $(m = 1)$, DMCTCP defaults to the standard TCP. However, when $m > 1$, there is a small difference on channels growth. The smallest channel can grow faster because it is least likely to cut rate as compared with other channels. Therefore, the algorithm simulates an increase parameter $\alpha = 2$ such that the increment of aggregated window w_{total} is always capped at $2/w_{min}$. For further discussion of the impact of these increase and decrease parameters, we will put that as one of our future work.

4.2. Implementation

We implemented DMCTCP by taking advantage of MPTCP API released in the Linux kernel 3.5.0 [26]. Each subflow in MPTCP is used as a channel in DMCTCP, which adopts the subsocket structure [34]. A master subsocket is responsible for upper layers, and it moves data among other subsockets as divide and conquer. The fast-retransmit and fast-recovery algorithms, as well as the Selective-Acknowledgment (SACK) option are inherited in each channel. The Smoothed-Round-Trip-Time ($srtt_i$) in each channel is used to estimate rtt. The number of channels per-flow can be configured statically or dynamically by a path manager. As mentioned in the algorithm, there is only one channel that follows slow start. After a congestion event

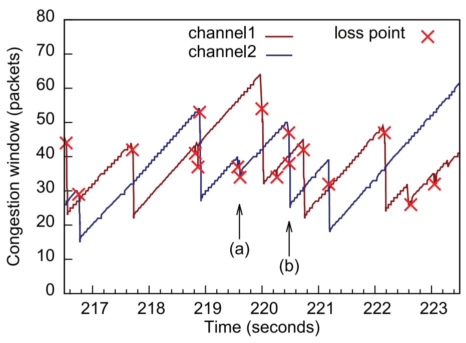

Fig. 3. Steady state behavior of a DMCTCP flow, $m = 2$.

happened, all other channels are added and they follow AIMD mechanisms with a minimum initial window defined in Linux.

4.3. Illustrative examples of DMCTCP

We developed a new packet-sniffing tool and implemented it over Linux network stack at kernel space [35]. By using this tool, we can clearly evaluate DMCTCP. As an example, Fig. 3 shows in stable phase each channel's congestion window with loss points (marked packet drop events) for a DMCTCP flow with two channels ($m = 2$). As can be seen from the figure, DMCTCP reacts to the two congestion types: (a) the smaller channel detects burst congestion, but the largest channel does not have any congestion at the time. So the smaller channel can merely fast retransmit the lost packets without cutting off its sending rate, (b) both the largest channel and the smaller channel detect congestions at the same time, but only the largest channel cuts its sending rate to half. Therefore, multiple channels consecutive cuts are avoided as desynchronization.

We also found that once loss synchronization is minimized, the convergence time of two competing flows is reduced. Fig. 4 illustrates this feature by comparing the convergence time between two consecutively started (30 s interval) flows. These two flows are competing a 100 Mb/s bandwidth 100 ms delay bottleneck. The routers have 128 packets in buffer size, which is roughly 15% BDP. The convergence time is defined to be the elapsed time when the congestion window of the second flow reaches roughly 80% of the first flow. For loss based congestion control algorithms, as shown in Fig. 4a and b, the convergence time of two TCP-SACK flows is almost 320 s and the convergence time of two CUBIC flows is almost 120 s. The pure delay based algorithm, TCP-Vegas [21], and the loss-delay based algorithm, TCP-Illinois [22], do not converge within the displayed 500 s as shown in Fig. 4c and d. This demonstrates the deficiency of queuing variance as a metric of congestion. However, when we compare the aggregated congestion window of each flow in DMCTCP, as shown in Fig. 4e and f, the convergence time is greatly reduced. It reduces more when the channel number increases.

In the same 100 Mb/s bandwidth 100 ms delay bottleneck with 15% BDP of buffer, we did another test to observe the congestion window responses to sudden arrived UDP traffic. Fig. 5 illustrates these congestion window responses of selected TCP variants to a 50 Mb/s UDP flow after 30 s. Because the UDP flow has a constant rate, we ignored it in Fig. 5. And since this bottleneck has more than 10% BDP buffer, we don't observe any burst congestion. As can be seen, all TCP variants start to converge to their fair share at time 30 s. Fig. 5a shows TCP-SACK has smaller bandwidth utilization than the others. Fig. 5b shows CUBIC have better bandwidth utilization than TCP-SACK, but it also has more packet drops. For a DMCTCP flow with two channels and three channels, Fig. 5c and d shows DMCTCP can efficiently utilize available bandwidth by regulating congestion window peaks, and these channels are very fair to each other. For a bandwidth utilization reference, the goodput of a 900 s running flow for TCP-SACK is 36.6 Mb/s, for CUBIC is 42.4 Mb/s, for MCTCP with two channels is 42.9 Mb/s, and for MCTCP with three channels is 43.7 Mb/s. These statistics show that TCP-SACK has the worst bandwidth utilization, although it has the least packet drops.

4.4. Determine a good number of channels

It is challenging to determine the best number of channels that can mitigate impacts of synchronization and burst congestion. On one hand, more channels can achieve better performance. On the other hand, a large number of channels render additional overhead such as channel states calculation and send/receive queue management. To

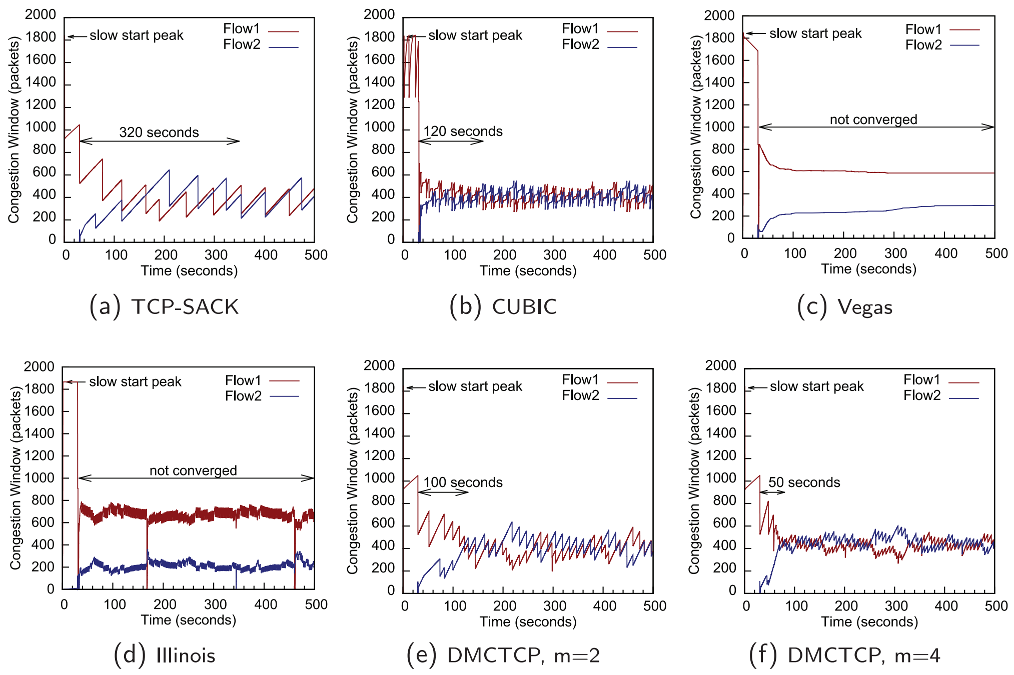

Fig. 4. Convergence time of two flows in a 100 Mb/s bandwidth, 100 ms delay bottleneck link.

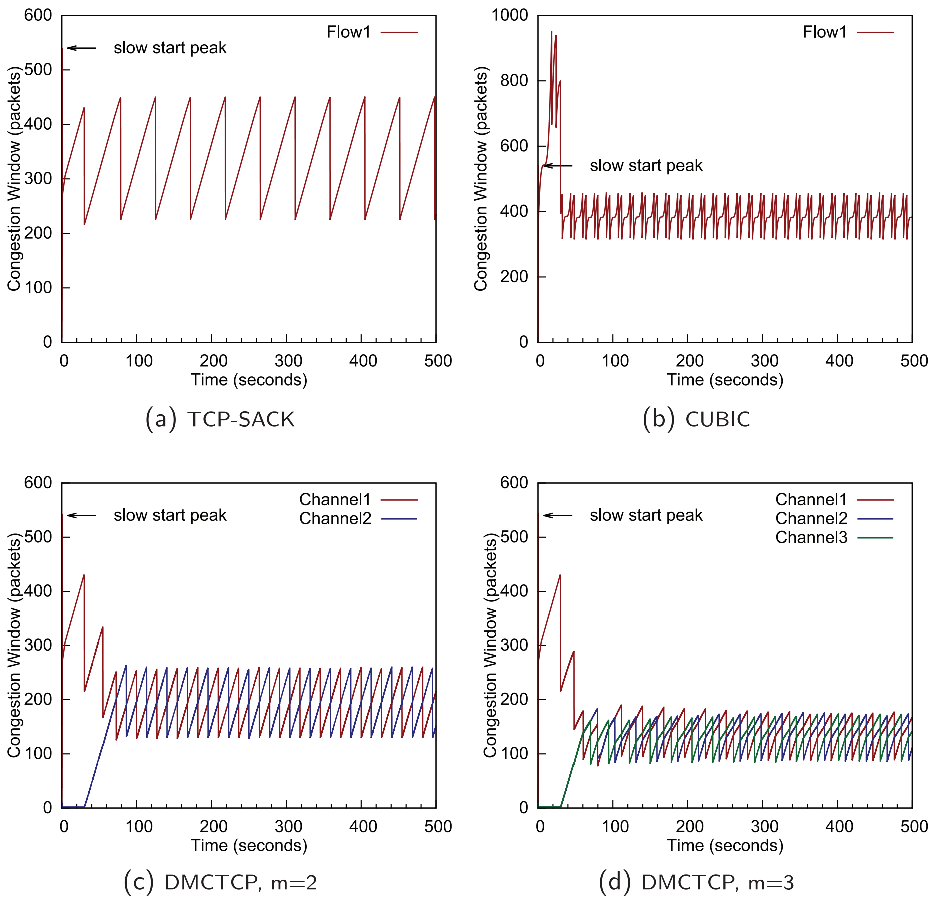

Fig. 5. Congestion window behavior of a TCP flow competing a 50 Mb/s UDP flow in a 100 Mb/s bandwidth, 100 ms delay bottleneck link.

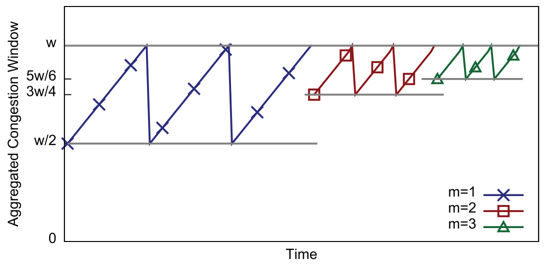

Fig. 6. The idealized aggregated window of various channels per-flow.

leverage performance and overhead, it is desirable to determine how many channels are good enough.

Ideally, the aggregated congestion window of a DMCTCP flow with complete desynchronization is expected to have regular variation between peaks and valleys. This is geometrically illustrated in Fig. 6, which shows the aggregated window of two and three channels per-flow, along with one channel per-flow as the standard TCP in model (2). Base on the geometric analysis of the sawtooth in Fig. 6, we find a simplified macroscopic model for a DMCTCP flow in steady-state[1]:

$$\text{average throughput} = \left(1 - \frac{\beta}{2 \cdot m}\right) \cdot C \qquad (3)$$

where β is the decrease parameter in each cut. Apply with $\beta = 1/2$ and enumerated m, we get the average link utilization in Table 1.

[1] The inference of this simplified throughput model is in Appendix A.

Table 1
Numerical examples of ideal link utilization of a DMCTCP flow.

Number of channels	Link utilization
$m = 1$	75%
$m = 2$	87.5%
$m = 3$	91.7%
$m = 4$	93.8%
$m = 5$	95%
$m = 6$	95.8%
$m = 7$	96.4%
$m = 8$	96.9%
...	
$m = 25$	99%

From Table 1, five channels per-flow can achieve 95% of link utilization. Beyond five channels, the increment is smaller. Consider the overhead of data reassembling and reordering, we recommend the channel number to be five.

4.5. TCP friendly

We measure the TCP-friendly of a flow variant over a TCP-SACK flow by calculating Jain's fairness index [36] from experiments with same RTT. These experiments use the same 100 Mb/s bandwidth 100 ms delay bottleneck link that is used in this section. Both flows start at same time and last 1200 seconds. The long term goodput of each flow is used for calculation of the index where 0 means the worst fairness and 1 means the best.

Fig. 7 shows the fairness indices as we vary the RTTs from 10 to 320 ms. As designed to be the same as the standard TCP, DMCTCP

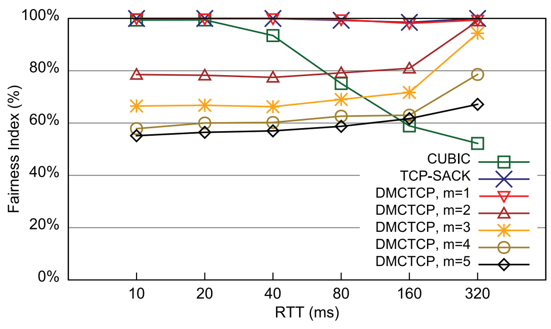

Fig. 7. TCP friendly comparison of TCP-SACK, CUBIC and DMCTCP.

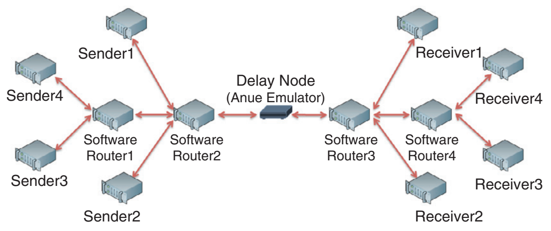

Fig. 8. 10 Gb/s access network testbed.

with one channel has the best TCP-friendliness. With more channels, DMCTCP becomes more aggressive. When the RTT increases, DMCTCP with more than two channels seems to be a little better TCP-friendly. We attribute this to the reason that slow convergence of TCP-SACK tends to be a little aggressive when it is competing with DMCTCP. We observe that in small BDP networks with RTT less than 40 ms, CUBIC also has very good TCP-friendliness. However, as BDP increases, CUBIC becomes more aggressive and it becomes the worst beyond 160 ms. As studied in 10 Gb/s experiments [37], the aggressiveness of CUBIC is expected in very large BDP networks because TCP-SACK is too conservative to be desirable in such networks. This opens a further discussion that if a network is TCP-friendliness sensitive, DMCTCP can be managed to use only one-channel per-flow.

5. Performance evaluation

5.1. Methodology

As shown in Fig. 8, we set up a 10 Gb/s dumbbell testbed as an example network from the CRON [38] platform. In this testbed, we evaluate the performance of DMCTCP and compare it with some other TCP variants.

Sender and receiver nodes are HP DL160G6 servers with two six-core 2.7 GHz processors and 16 GB RAM. Router nodes are SUN Fire x4240 servers with two quad-core 2.7 GHz processors and 8 GB RAM. All nodes have Myricom 10Gb/s NICs such that the bottleneck link rate confirms to be 10 Gb/s. The delay node in the middle is an Anue XGEM optical hardware emulator [39]. This emulator can provide bi-directional communication delay and additional packet drop rate.

All the nodes use Ubuntu Server 12.04. Each end-host has a MPTCP enabled Linux 3.5.0 kernel to test TCP-SACK, CUBIC [24], MPTCP-Coupled [26], MPTCP-OLIA [27] and our implementation of DMCTCP. These end-hosts have enabled Interrupt-Coalescing and TCP Segmentation-Offloading, and are configured to have a very large buffer such that the sending rates of flows are limited only by the congestion control algorithm. The routers use a normal Linux 3.5.0 kernel with the latest NIC driver.[2] FIFO drop-tail queuing policy is configured in each router.

[2] Driver version: myri10ge-Linux.1.5.3devel-march-2013.

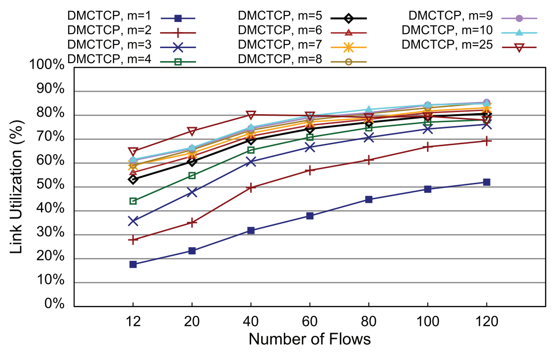

Fig. 9. Average bottleneck link utilization as a function of flows without background traffic, compared with different number of channels per-flow in DMCTCP.

Because we use an emulation-based testbed, it is important to make our network similar to the real network that has tiny buffers. We empirically found that more system tunings at data link layer and network layer are necessary. To implement a tiny buffer, we need to reduce both the *tx_ring*/*rx_ring* at layer 2 in NIC and *netdev_backlog*/*qdisc* at layer 3 in kernel. We modified the NIC driver to use only 128 ring descriptors, a small portion of the default 1024 size. Then, we set packet size (MTU) to 8000 bytes because each descriptor holds up to 4096 bytes of data. This means a packet requires two descriptors. We also set *netdev_backlog*/*qdisc* to 2. Therefore, our configuration emulates a 10 Gb/s network environment with 66 packets of tiny buffers.

There are four paths from $Sender_i$ to $Receiver_i$. We use Iperf traffic generator to generate long-lived TCP flows. We also evaluate our experiments performed in the context of bursty background traffic, which is injected by Router1 and Router4. Two types of flows are considered as the background traffic: short-lived TCP flows and real-time UDP flows. We use Harpoon traffic generator to infinitely transmit short-lived TCP flows with an average delay of 120 ms. The inter-arrival times between two successive connections follow exponential distribution with mean 1 s. The average file size is 1 MB from a total of 200,000 randomly sorted files, which follows Pareto distribution with the shape parameter alpha = 1.5. These values are realistic, based on comparisons with actual packet traces [6]. The aggregated throughput of short-lived TCP traffic is averaged at 165 Mb/s, along with one additional 300 Mb/s UDP flow as an aggregate of many individual real-time streams. As a result, the average background traffic is 4.6% of the bottleneck link capacity.

5.2. Results[3]

5.2.1. Average bottleneck link utilization

The average bottleneck link utilization is calculated by counting goodputs of all flows. Bi-directional delay (RTT) of each path is fixed at 60 ms, 120 ms, 180 ms and 240 ms. Long-lived TCP flows are sent out in same number. These flows start within [0,10] seconds. One trial for each test lasts 900 seconds, and repeats three times to get arithmetic means of the goodputs. Most standard deviations fall in the range of ± 4% such that they can be omitted in our results.

We first verify the performance of DMCTCP by increasing the number of channels per-flow. Fig. 9 shows the average bottleneck link utilization without background traffic. It shows that utilization gets higher when the number of channels increases. Starting with five channels per-flow, nearly 80% link utilization is achieved with only 100 flows. When the number of channels increases, the increment of utilization becomes smaller. Therefore, it matches our analysis of

[3] Readers can find the results of Intra-protocol fairness and RTT fairness from our previous conference paper [35].

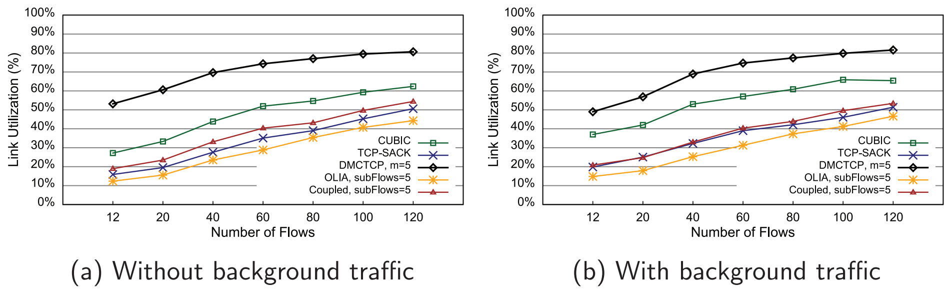

Fig. 10. Average bottleneck link utilization as a function of flows, compared with different TCP variants.

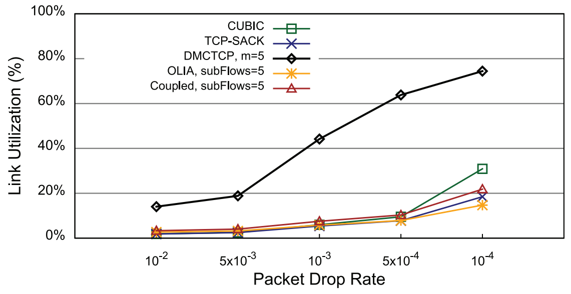

Fig. 11. Average bottleneck link utilization of 20 flows as a function of packet drop rate, RTT = 60 ms.

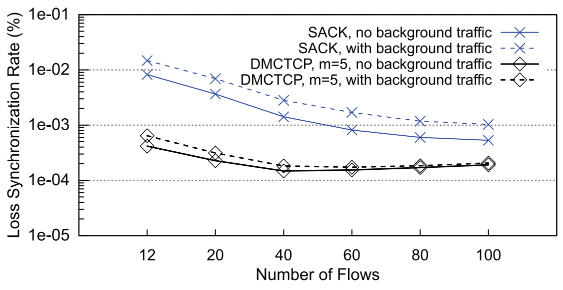

Fig. 12. Global TCP loss synchronization rates of TCP-SACK and DMCTCP (m = 5).

Table 2
Goodput ratio of other TCP vairants over TCP-SACK with different packet drop rates.

Goodput ratio	10^{-2}	5×10^{-3}	10^{-3}	5×10^{-4}	10^{-4}
DMCTCP	**761%**	**757%**	**818%**	**819%**	**403%**
CUBIC	102%	102%	110%	122%	167%
MPTCP-coupled	180%	162%	139%	132%	118%
MPTCP-OLIA	146%	128%	107%	101%	80%

determining five channels per-flow. It also shows that the utilization of 25 channels per-flow starts to be downgraded beyond 60 flows. This downgrade is likely caused by the overhead of data reassembling at the receiver side.

Secondly, we compare the performance of DMCTCP ($m = 5$) with other TCP variants without and with background traffic. As shown in Fig. 10a and b, DMCTCP performs best. Compared with TCP-SACK, the utilization is 60% to 200% higher. Compared with CUBIC, the utilization is 30% to 80% higher. Although we set five subflows per-flow for MPTCP-Coupled and MPTCP-OLIA, they have close performance to TCP-SACK. This verifies TCP-SACK, MPTCP-Coupled and MPTCP-OLIA are conservative under large BDP networks. As shown in Fig. 10b, with background traffic, the link utilizations are a little higher than these without background traffic. This is expected because the background traffic introduces some dynamics on queuing occupancy as in studies [37,40].

5.2.2. Adding packet drop rate

In study [2], network traffic is in favor of Poisson process and the loss probability of a tiny-sized buffer running at 95% utilization is approximately 3.3×10^{-4}. Therefore, in the bottleneck, we added an additional packet drop rate that follows Poisson process provided by the hardware emulator [39]. We calculated the average bottleneck link utilization by aggregating goodputs of 20 long-lived TCP flows with 60 ms RTT. Again, one trial for each test lasts 900 seconds and is repeated three times to get arithmetic means. The standard deviations are small enough (less than 2%) to be omitted.

As shown in Fig. 11, DMCTCP has much better performance than the other TCP variants. We also assess performance of TCP variants by showing the ratio of the achieved throughput of the TCP variant with the achieved throughput of TCP-SACK. Table 2 summarizes these throughput ratios. We can observe that DMCTCP performance is more robust in high loss rate networks.

5.2.3. TCP loss synchronization rate

To better understand the impact of TCP loss synchronization, we define new formulas of calculating the per-flow or per-channel synchronization rate and the global synchronization rate.

We specify a congestion event happens when there are one or more packets dropped simultaneously within a "short" interval Δt. In the time domain, Δt is roughly the largest *rtt* of these flows. There are two separated congestion events if their Δts are not overlapped. For a total of n flows through a bottleneck over a "long" measuring period τ, we let T denote the total number of congestion events, where $l_{i,k}$ represents a loss event at the kth congestion for flow i such that $l_{i,k} = 1$ when flow i loses, and $l_{i,k} = 0$ otherwise. Therefore, the total number of loss events N_i for flow i is: $\sum_{k=1}^{T} l_{i,k} = N_i \in [1, \ldots, T]$. These dropped packets are either from a single flow or from multiple flows. The per-flow or per-channel synchronization rate is defined in Formula (4), which is based on the *weighted* loss events of a specific flow or channel i over the total congestion events T:

$$SR_i = \frac{N_i^w}{T} = \frac{1}{T}\sum_{k=1}^{T}(l_{i,k} \times weight_k) \qquad (4)$$

As mentioned above, the $l_{i,k}$ equals 1 when flow i or channel i cuts off at the kth congestion event, and the total number of weighted loss events N_i^w of flow i or channel i equals $\sum_{k=1}^{T}(l_{i,k} \times weight_k)$ where the weight ($weight_k$) is calculated based on results from our packet-sniffing tool. For example, 3 out of total n flows or $n \times m$ channels are loss synchronized on the kth congestion so the corresponding $weight_k$ is $3/n$ or $3/(n \times m)$.

To average n flows or $n \times m$ channels synchronization rates, a global TCP synchronization rate is expressed below as a harmonic mean of them in Formula (5). The n' equals n for flows or $n \times m$ for

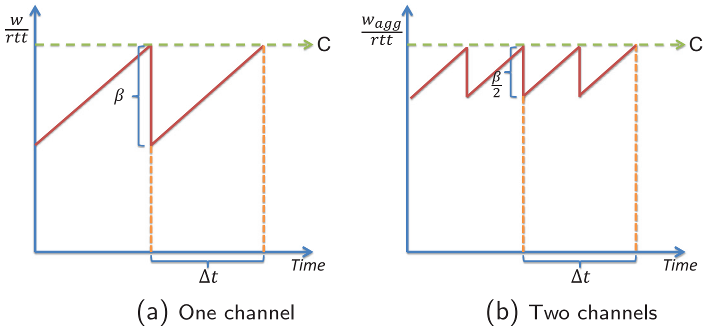

Fig. A.13. Congestion control scenarios of a DMCTCP flow with two channels.

channels.

$$\overline{SR} = n' / \left(\sum_{i=1}^{n'} \frac{1}{SR_i} \right) \tag{5}$$

We compare the loss synchronization rates of TCP-SACK and DMCTCP because they closely follow the AIMD policy. Using our TCP stack sniffing tool, we tested various number of flows without background traffic and with background traffic. Bi-directional delay between $Sender_i$ and $Receiver_i$ are fixed at 120 ms, and same number of long-lived TCP flows are sent out from $Sender_i$ to $Receiver_i$ respectively for 1500 s. After a warm up of 300 s, we calculate TCP loss synchronization rates for 1000 s.

Fig. 12 shows the global TCP loss synchronization rates as a function of different number of flows or channels. Compared with TCP-SACK, DMCTCP ($m = 5$) has synchronization rates roughly one order of magnitude smaller, which shows that it can desynchronize channels for better performance. We notice that synchronization rates of DMCTCP remain nearly the same beyond 40 flows and the rates of TCP-SACK are a little higher for cases under background traffic. The reason for the first could be the bottleneck queue causes more losses as the number of flows increases. And the reason for the second is probably due to the dynamics of background traffic are taking penalties that help increase loss synchronization on long-lived flows.

6. Conclusion

We have shown that popular TCP variants have performance issues in high speed networks with tiny buffers. Based on the analyses, DMCTCP tries to avoid TCP loss synchronization and to avoid impact of burst congestion. Our experimental results confirmed that DMCTCP can achieve much higher performance with very good properties in terms of convergence, TCP-friendliness and fairness.

DMCTCP brings immediate benefits as it does not require modifications to or assistance from other layers and it matches the requirements to deploy all-optical routers in access networks. Because queuing delay is minimized, it also significantly reduces the memory required for maintaining a large window and is beneficial to latency sensitive applications.

Acknowledgment

This work has been supported in part by the NSF CC-NIE grant (NSF award #1341008).

Appendix A. Inference of DMCTCP throughput model

As shown in Fig. A.13a, for a single-channel DMCTCP flow's congestion window W in steady-state, if the network buffer depth is negligible, RTT is almost a fixed value and W will be cut in half ($\beta = 1/2$) when the sending rate reaches the bottleneck link capacity C. Then it linearly increases by one segment per-RTT until again reaches W. Therefore, the total data transmitted during the time Δt is simplified as:

$$\text{average throughput} \cdot \Delta t = C \cdot \Delta t - 1/2 \cdot \beta \cdot C \cdot \Delta t$$

$$\Rightarrow \text{average throughput} = \left(1 - \frac{\beta}{2}\right) \cdot C$$

Ideally, the aggregated congestion window of a DMCTCP flow with complete desynchronization is expected to have regular variations. This is geometrically illustrated in Fig. A.13b, which shows the aggregated window of two channels. The total data transmitted during the time Δt is simplified as:

$$\text{average throughput} \cdot \Delta t = C \cdot \Delta t - 2 \cdot 1/2 \cdot \frac{\beta}{2} \cdot C \cdot \frac{\Delta t}{2}$$

$$\Rightarrow \text{average throughput} = \left(1 - \frac{\beta}{4}\right) \cdot C$$

For a DMCTCP flow with m channels, therefore, it will have a fine-grained aggregated congestion window that is desynchronized by these channels. The total data transmitted during the time Δt is simplified as:

$$\text{average throughput} \cdot \Delta t = C \cdot \Delta t - m \cdot 1/2 \cdot \frac{\beta}{m} \cdot C \cdot \frac{\Delta t}{m}$$

$$\Rightarrow \text{average throughput} = \left(1 - \frac{\beta}{2 \cdot m}\right) \cdot C$$

References

[1] C. Villamizar, C. Song, High performance TCP in ANSNET, ACM SIGCOMM Comput. Commun. Rev. 24 (5) (1994) 45–60.
[2] D. Wischik, N. McKeown, Part I: buffer sizes for core routers, ACM SIGCOMM Comput. Commun. Rev. 35 (3) (2005) 75–78.
[3] G. Raina, D. Towsley, D. Wischik, Part II: control theory for buffer sizing, ACM SIGCOMM Comput. Commun. Rev. 35 (3) (2005) 82.
[4] G. Raina, D. Wischik, Buffer sizes for large multiplexers: TCP queueing theory and instability analysis, in: Proceedings of the Next Generation Internet Networks, 2005, IEEE, 2005, pp. 173–180.
[5] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, Part III: routers with very small buffers, ACM SIGCOMM Comput. Commun. Rev. 35 (3) (2005) 83–90.
[6] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, Experimental study of router buffer sizing, in: Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement, ACM, 2008, pp. 197–210.
[7] H. Park, E.F. Burmeister, S. Bjorlin, J.E. Bowers, 40-GB/s optical buffer design and simulations, Numer. Simul. Optoelectron. Devices (NUSOD) (2004).
[8] H. Sivakumar, S. Bailey, R.L. Grossman, Psockets: the case for application-level network striping for data intensive applications using high speed wide area networks, in: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2000, p. 37.
[9] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, Architectural guidelines for multipath tcp development, RFC6182 (March 2011) (2011), www.ietf.ort/rfc/6182.
[10] S. Floyd, Tools for the evaluation of simulation and testbed scenarios, 2008, http://tools.ietf.org/html/draft-irtf-tmrg-tools-05.
[11] Q. Fu, P. Jay, A step towards understanding loss synchronisation between concurrent TCP flows, in: Proceedings of the INFOCOM Workshops 2008, IEEE, IEEE, 2008, pp. 1–6.

[12] S. Hassayoun, D. Ros, Loss synchronization and router buffer sizing with high-speed versions of TCP, in: Proceedings of the INFOCOM Workshops 2008, IEEE, IEEE, 2008, pp. 1–6.
[13] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, M. Yasuda, Less is more: trading a little bandwidth for ultra-low latency in the data center, in: Proceedings of the USENIX NSDI Conference, 2012.
[14] Y. Cai, B. Jiang, T. Wolf, W. Gong, A practical on-line pacing scheme at edges of small buffer networks, in: Proceedings of the INFOCOM, 2010, IEEE, IEEE, 2010, pp. 1–9.
[15] A. Vishwanath, V. Sivaraman, M. Thottan, C. Dovrolis, Enabling a bufferless core network using edge-to-edge packet-level FEC, in: Proceedings of the INFOCOM, 2010, IEEE, IEEE, 2010, pp. 1–9.
[16] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), ACM SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 63–74.
[17] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, Routers with very small buffers, in: Proceedings of the IEEE Infocom, vol. 6, Citeseer, 2006.
[18] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Trans. Netw. 1 (4) (1993) 397–413.
[19] S. Hassayoun, D. Ros, Loss synchronization, router buffer sizing and high-speed TCP versions: adding red to the mix, in: Proceedings of the 34th Conference on Local Computer Networks, 2009. LCN 2009. IEEE, IEEE, 2009, pp. 569–576.
[20] C. Jin, D. Wei, S. Low, Fast TCP: motivation, architecture, algorithms, performance, in: Proceedings of the 23rd AnnualJoint Conference of the IEEE Computer and Communications Societies INFOCOM 2004., vol. 4, IEEE, 2004, pp. 2490–2501.
[21] L. Brakmo, L. Peterson, TCP vegas: end to end congestion avoidance on a global internet, IEEE J. Selected Area. Commun. 13 (8) (1995) 1465–1480.
[22] S. Liu, T. Başar, R. Srikant, Tcp-illinois: a loss-and delay-based congestion control algorithm for high-speed networks, Perform. Eval. 65 (6) (2008) 417–440.
[23] K. Tan, J. Song, A compound tcp approach for high-speed and long distance networks, in: Proceedings of the IEEE INFOCOM, Citeseer, 2006.
[24] S. Ha, I. Rhee, L. Xu, Cubic: a new TCP-friendly high-speed TCP variant, ACM SIGOPS Oper. Syst. Rev. 42 (5) (2008) 64–74.
[25] T.J. Hacker, B.D. Athey, B. Noble, The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network, in: Proceedings of the International Parallel and Distributed Processing Symposium International, IPDPS 2002, Abstracts and CD-ROM, IEEE, 2001, pp. 10–pp.
[26] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, Design, implementation and evaluation of congestion control for multipath TCP, in: Proceedings of the 8th USENIX Conference on Networked systems design and implementation, USENIX Association, 2011, p. 8.
[27] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.-Y. Le Boudec, MPTCP is not pareto-optimal: performance issues and a possible solution, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, ACM, 2012, pp. 1–12.
[28] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, Is it still possible to extend TCP? in: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, ACM, 2011, pp. 181–194.
[29] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, in: Proceedings of the ACM SIGCOMM Computer Communication Review, vol. 41, ACM, 2011, pp. 266–277.
[30] J. Mahdavi, S. Floyd, Tcp-friendly unicast rate-based flow control, 1997, ftp://ftp.cs.umass.edu/pub/net/cs691q/tcp-friendly.txt. Technical note sent to the end2end-interest mailing list.
[31] J.F. Kurose, K.W. Ross, Computer Networking: A Top-Down Approach, 6th edition, Pearson, pp. 278–279.
[32] L. Xue, S. Kumar, C. Cui, P. Kondikoppa, C.-H. Chiu, S.-J. Park, Afcd: an approximated-fair and controlled-delay queuing for high speed networks, in: Proceedings of the 22nd International Conference on Computer Communications and Networks (ICCCN), 2013, IEEE, 2013, pp. 1–7.
[33] B. Shihada, P.-H. Ho, Transport control protocol in optical burst switched networks: Issues, solutions, and challenges, Commun. Surv. Tutorials IEEE 10 (2) (2008) 70–86.
[34] C. Raiciu, C. Paasch, S. Barr, A. Ford, M. Honda, F. Duchene, O. Bonaventure, M. Handley, How hard can it be? designing and implementing a deployable multipath tcp, in: Proceedings of the USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA), 2012.
[35] C. Cui, L. Xue, C.-H. Chiu, P. Kondikoppa, S.-J. Park, DMCTCP: Desynchronized multi-channel TCP for high speed access networks with tiny buffers, in: Proceedings of the 23nd International Conference on Computer Communications and Networks (ICCCN), 2014, IEEE, 2014, pp. 1–8.
[36] R. Jain, A. Durresi, G. Babic, Throughput Fairness Index: An Explanation, Technical Report, Tech. rep., Department of CIS, The Ohio State University, 1999.
[37] L. Xue, S. Kumar, C. Cui, S.-J. Park, An evaluation of fairness among heterogeneous TCP variants over 10GBps high-speed networks, in: Proceedings of the 37th Annual IEEE Conference on Local Computer Networks (LCN 2012), IEEE, 2012, pp. 344–347.
[38] CRON, CRON Project: Cyberinfrastructure for Reconfigurable Optical Networking Environment, 2011, http://www.cron.loni.org/.
[39] Ixia, Anue XGEM 10Gig Ethernet Emulator, 2013, http://www.ixiacom.com/pdfs/datasheets/gem_xgem.pdf.
[40] S. Ha, L. Le, I. Rhee, L. Xu, Impact of background traffic on performance of high-speed TCP variant protocols, Comput. Netw. 51 (7) (2007) 1748–1762.