TCon: A Transparent Congestion Control Deployment Platform for Optimizing WAN Transfers

Yuxiang Zhang[1], Lin Cui[14*], Fung Po Tso[2], Quanlong Guan[1] and Weijia Jia[3]

[1]Department of Computer Science, Jinan University, Guangzhou, P.R.China
[2]Department of Computer Science, Loughborough University, UK
[3]Department of Computer Science & Engineering, SJTU, P.R.China
[4]Guangdong Key Laboratory of Big Data Analysis and Processing, P.R.China
`samuelzyx0924@gmail.com; tcuilin@jnu.edu.cn;`
`p.tso@lboro.ac.uk; gql@jnu.edu.cn; jia-wj@cs.sjtu.edu.cn`

Abstract. Nowadays, many web services (e.g., cloud storage) are deployed inside datacenters and may trigger transfers to clients through WAN. TCP congestion control is a vital component for improving the performance (e.g., latency) of these services. Considering complex networking environment, the default congestion control algorithms on servers may not always be the most efficient, and new advanced algorithms will be proposed. However, adjusting congestion control algorithm usually requires modification of TCP stacks of servers, which is difficult if not impossible, especially considering different operating systems and configurations on servers. In this paper, we propose *TCon*, a light-weight, flexible and scalable platform that allows administrators (or operators) to deploy any appropriate congestion control algorithms transparently without making any changes to TCP stacks of servers. We have implemented *TCon* in Open vSwitch (OVS) and conducted extensive test-bed experiments by transparently deploying BBR congestion control algorithm over *TCon*. Test-bed results show that the BBR over *TCon* works effectively and the performance stays close to its native implementation on servers, reducing latency by 12.76% on average.

Keywords: Congestion Control, BBR, Transparent

1 Introduction

Recent years, many web applications have moved into cloud datacenters to take advantage of the economy of scale. Since bandwidth remains relatively cheap, web latency is now the main impediment to improving service quality. Moreover, it is well known that web latency inversely correlates with revenue and profit. For instance, Amazon estimates that every 100ms increase in latency cuts profits by 1% [7]. Reducing latency, especially the latency between datacenter and clients through WAN environment, is of primary importance for providers.

* Corresponding author: Lin Cui (tcuilin@jnu.edu.cn)

In response, administrators adopt network appliances to reduce network latency. For example, TCP proxies and WAN optimizers are used for such optimization [5][9]. However, when facing dramatically increasing traffic, they would degrade performance [5]. Furthermore, the split-connection approach used in TCP proxy would break a TCP connection into several sub-connections, destroying TCP end-to-end semantics. Applications may receive an ACK for the data which are actually still in transmitting, potentially violate the sequential processing order [3][6]. On the other hand, WAN optimizers perform compression on data which may add additional latency and require additional decompression appliances in ISPs for decompressing data from optimizers[3].

In addition to using network appliances, enhancement of TCP congestion control is considered to reduce latency, since most web services use TCP. Many TCP congestion control algorithms have been proposed, e.g., Reno [8], CUBIC [10] and recent BBR [4]. These proposals perform very well in their target scenarios while have performance limitations when working under different circumstance. And the degradation of performance caused by in-appropriate congestion control algorithms can lead to loss and increased latency. Furthermore, cloud datacenters have many web servers[1] which have different operating systems and configurations, e.g., Linux or Windows with different kernel versions and congestion control algorithms. Considering such diversity and large number of web servers in cloud datacenters, adjusting congestion control algorithms (e.g., deploying new advanced algorithms) is a daunting, if not impossible, task. Hence, a question arise in our mind: *Can we find a way to transparently deploy advanced congestion control without modifying TCP stacks of web servers?*

In this paper, we present *TCon*, a Transparent Congestion control deployment platform without requiring changing TCP stack of servers. *TCon* can implement target TCP congestion control within Open vSwitch (OVS) to reduce the latency of WAN transfers. At a high-level (as illustrated in Figure 1), *TCon* monitors packets of a flow through OVS and modifies packets to reconstruct important TCP parameters for congestion control (e.g., *cwnd*). *TCon* runs congestion control specified by administrators and then forces intended congestion window by modifying the receive window (*rwnd*) on incoming ACKs.

The main contributions of this paper are as follows:

1. We designed a transparent platform *TCon*, which allows network administrators (or operators) deploying new advanced congestion control algorithms without modifying TCP stacks of servers.
2. A prototype of *TCon* is implemented based on OVS. *TCon* is light-weight, containing only about 100 lines of code. BBR congestion control algorithm is implemented on *TCon* as an example, using around 900 lines of code.
3. Extensive test-bed experiments are conducted, including WAN connections from both Shanghai and Oregon. Experiment results show that *TCon* works effectively, reducing latency by 12.76% on average.

[1] Those web servers can be either physical servers or VMs in cloud datacenters. For consistency, we use “web server” to refer both cases.

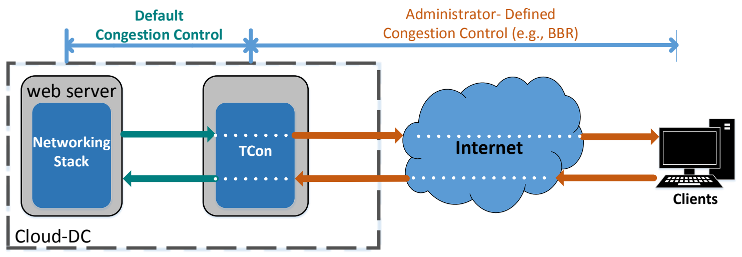


Fig. 1. High-level illustration of TCon.

2 Motivations and Objectives

2.1 Motivations

Latency is important for web service which is closely linked to revenue and profit [7]. Large latency degrades service performance, resulting in worsening customer experience and hence revenue loss. In light of this, it is always in service providers' interest to minimize the latency.

Many service providers use network functions such as TCP proxies and WAN optimizers to improve the latency performance [5][9]. TCP proxy can quickly prefetch packets from servers at a high rate, then send them to the destination using several sub-connections. While WAN optimizer performs compression and caching operation on data for shaping network traffic. However, the scalability of these network functions is a great challenge. When there is a burst of requests for service, the performance of such network functions can be easily saturated due to the insufficient processing capacity. Moreover, TCP proxy goes against TCP end-to-end semantics. For instance, a barrier-based application may believe that all its packets were ACKed, and advance to the next phase, while they were not actually received, potentially causing errors in the application [3][6]. Furthermore, WAN optimizer adopts compression to speed up transfers but this may add additional latency on service and require ISPs to offer co-operating decompression appliances [3].

On the other hand, TCP congestion control algorithm is known to significantly impact network performance. As a result, TCP congestion control has been widely studied and many schemes have been proposed to improve performance [4][8][10]. These schemes perform well in their own target scenarios while get limiting performance in other circumstance. However, service providers usually deploy a diverse of web servers which may run on different versions of operating systems (e.g., Linux and Windows) and be configured with different congestion control algorithms. Adjusting TCP stacks of such large amount of web servers is a daunting task, if not impossible. Furthermore, in multi-tenants cloud datacenters, network operators may be prohibit from upgrading TCP stacks of web servers for security issues. Therefore, *significant motivations exist to deploy advanced congestion control algorithms (e.g., BBR) transparently* .

2.2 Objectives of *TCon*

The goal of *TCon* is to provide a transparent platform allowing network administrators to deploy new advanced congestion algorithms without modifying TCP stacks of web servers. A number of *TCon*'s characteristics led to our design approaches are summarized as follows:

1. **Transparency**. *TCon* allows network administrators or operators to enforce advanced congestion control algorithms without touching TCP stacks of servers. Deployment and operations of *TCon* should be transparent to both web servers and clients. This is important in untrusted public cloud environments or simply in cases where servers cannot be updated due to a dependence on a specific OS or library [13].
2. **Flexibility**. *TCon* allows different congestion control algorithms to be applied on a per-flow basis. This is useful because each congestion control algorithm has its own deficiency and suitable scenarios. Allowing adjusting congestion control algorithms on a per-flow basis can enhance flexibility and performance.
3. **Light-weight**. While the entire TCP stack may seem complicated and prone to high overhead, the congestion control aspect of TCP is relatively light-weight and simple to implement. Indeed, the prototype implementation of *TCon* on OVS has around 100 lines of code for basic functionalities. And the BBR algorithm over *TCon* contains about 900 lines of code.

2.3 BBR congestion control

Recently, BBR is proposed in [4], which adopts a novel control window computation algorithm based on bandwidth delay product (BDP). In BBR, sender needs to continuously estimate the bottleneck bandwidth (BtlBw) and round-trip propagation time (RTprop) and let the total data in flight be equal to the BDP (= BtlBw $\times$ RTprop). By adjusting the *cwnd* (otherwise the sending rate) based on BDP, BBR guarantees that the bottleneck can run at 100 percent utilization and preventing bottleneck starvation but not overfilling. Therefore, BBR can keep low latency since its *cwnd* is not based on loss but network capacity. Meanwhile, loss and RTT fluctuations are not rare in WAN. However, loss is not considered as congestion signals in BBR, which uses RTprop as the metric of network capacity to get rid of RTT fluctuations caused by queuing delay [4].

Because of those reasons above, BBR is suitable for most WAN environment. As a case study, we will deploy BBR over *TCon* to demonstrate its effectiveness.

3 Design and Implementation

This section provides an overview of *TCon*'s design details. For simplicity, we use BBR as an example for explanation. Other congestion control algorithms can also be easily implemented on *TCon* similarly.

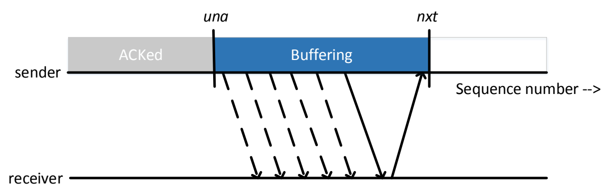


Fig. 2. Variables for TCP sequence number space.

3.1 Obtaining Congestion Control State

Since *TCon* is implemented in the *datapath* of the OVS (illustrate in Section 3.3), all traffic can be monitored. We now demonstrate how congestion control state (e.g., RTprop and BtlBw) can be inferred on packet level.

Figure 2 provides a visual of TCP sequence number space. The *una* is the first packet's sequence number which has been sent, but not yet ACKed. The *nxt* is the sequence number of the next packet to be sent (but *TCon* hasn't received it yet). Packets between *una* and *nxt* are inflight. Variable *una* is simple to update: each ACK contains an acknowledgement number (*acknum*), and *una* is updated when $acknum > una$. When a packet is received from web servers, *nxt* is updated if its sequence number is larger than or equal to current value of *nxt*.

Measuring RTprop is relatively simple. The arriving timestamps of each packet is recorded. When ACK arrives, RTT is obtained by computing the difference between ACK and corresponding arriving timestamps. The minimal RTT in a short period is regarded as RTprop [4]. BtlBw can be estimated by monitoring the delivery rate. The *delivered* variable is the delivery size between conjoint ACK which can be measured by counting being acknowledged packets' size. Then delivery rate can be inferred as dividing *delivered* by the difference between conjoint ACK's arriving timestamps. BtlBw is the maximal delivery rate in a short period [4]. Detecting packet loss is also relatively simple. If $acknum \leq una$, the a local *dupack* counter is updated. When *dupack* counts to 3, it means a packet loss happened [12].

3.2 Enforcing Congestion Control

The basic parameters of BBR, e.g., BtlBw and RTprop, can be tracked as described in Section 3.1 for each connection. Then the sending rate is computed by multiplying BtlBw and RTprop, which is translated into window size later, i.e., *cwnd*. Our implementation closely tracks the Linux source code of BBR and more details can be found in [4].

Moreover, there must be a mechanism to ensure a web server's TCP flow adheres to the window size determined in the *TCon*. Luckily, TCP provides built-in functionality that can be reprovisioned for *TCon*. Specifically, TCP's flow control allows a receiver to advertise the amount of data that it is willing to process via a receive window (*rwnd*) [12]. Ensuring a web server's flow adheres to *rwnd* is relatively simple. The *TCon* computes a new congestion window *cwnd* every time an ACK is received, which provides an upper bound on the

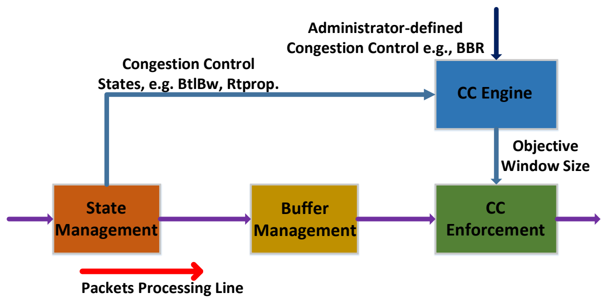


Fig. 3. The architecture of *TCon*.

number of bytes the web server's flow is now able to send. If it is smaller than the packets' original *rwnd*, *TCon* overwrites *rwnd* with its computed *cwnd*, i.e., *rwnd*=min(*cwnd*, *rwnd*). Such scheme restricts amount of packets sent from server to clients while preserving TCP semantics. Web servers with unalterd TCP stacks will naturally follow the standard.

Besides, WAN always has high packet loss rate. However, web server's default congestion control is usually a loss sensitive scheme which would aggressively reduces *cwnd* when receiving loss signal. In order to prevent *TCon*'s sending rate from throttling by the web server, *TCon* buffers all packets which restricts the size of buffering size less than the computed window size and retransmit the loss rather than web server does it. Meanwhile, *TCon* handles all congestion signals (e.g., ECN feedback and three duplicated ACKs), preventing them from reaching to web servers, which would reduce *cwnd* of web servers.

3.3 Implementation

We have implemented *TCon* on Open vSwitch (OVS) v2.6.0 [1]. About 100 lines of code are used to implement *TCon*'s basic functions, e.g., obtaining congestion control states and managing buffer. Another 900 lines of code are for the implementation of BBR on *TCon*. Flows are hashed on a 5-tuple (IP addresses, ports and protocol) to obtain a flow's state which is used to maintain the congestion control state mentioned in Section 3.1. SYN packets are used to create flow entries, and FIN packets are used to remove flows entries. Other TCP packets, such as data and ACKs, trigger updates to flow entries. Since there are many more table lookup operations (to update flow state), Read-Copy-Update (RCU) hash tables are used to enable efficient lookups. Additionally, individual *spinlocks* are used on each flow entry in order to allow for multiple flow entries to be updated simultaneously. Furthermore, *skb_clone*() is used for packet buffering to prevent deep-copy of data.

Finally, the overall architecture of *TCon* is shown in Figure 3. A web server generates a packet that is pushed down the network stack to OVS. The packet is intercepted in *ovs_dp_process_packet*(), where packet's flow entry is obtained by *StateManagement*. Sequence number state is updated and the sending timestamps are recorded. Then these packets are buffered by *BufferManagement* and sent to clients. When ACKs eventually from clients reach *TCon*, *CCEngine* module uses the congestion control states offered by *StateManagement* to compute a new congestion window. Then *CCEnforcement* module modifies *rwnd* if needed and recomputes the checksum before pushing the packet to the network.

3.4 Deployment locations of *TCon*

Since *TCon* is implemented on OVS, it can be easily deployed in three possible locations in cloud datacenter:

- *VMs*: Deploying *TCon* in VMs allows network administrators to setup new *TCon* servers or release old ones dynamically for load-balancing. However, such scheme requires routers/switches redirecting desired traffic to *TCon* servers, which is not difficult specially for SDN-enabled environment.
- *Hypervisors*: As OVS is compatible with most hypervisors, *TCon* can also be deployed in hypervisors of physical servers. Such scheme allows *TCon* to be easily scaled with number of servers in datacenters. It also minimizes the latency between *TCon* and web servers, i.e., VMs. Furthermore, no route redirection is required in this case. However, the flexibility and scalability are limited considering migrations of VMs or situation that VMs on a server are heavy loaded.
- *Routers/Switches*: *TCon* can also be deployed with OVS on routers/switches in datacenters. Routers/switches can inherently monitoring all incoming traffic, making *TCon* can easily enforce congestion control without route redirection. However, traffic sent through a router/switch is determined by the routing algorithm of datacenters, and it is difficult to perform load balancing. And heavy traffic may also overwhelm capacity of routers/switches.

Each deployment choice is suitable for different requirements and scenarios. In our current implementation, *TCon* is deployed on VMs in datacenter. In practice, combination of these three deployment choices above can be considered.

4 Evaluation

4.1 Experiment Setup

We have deployed three VMs (*TCon* and two web servers, see Table 1) in the campus datacenter (located in Jinan University, Guangzhou China), which contains over 400 web servers (VMs) running on 293 physical servers. The bandwidth of the datacenter to the Internet is about 20Gbps, shared by all servers in the datacenter. And the data rate of NIC on each physical server is 1Gbps, shared by all VMs on the server.

The BBR congestion control algorithm is implement over *TCon*. The baseline scheme, CUBIC, is Linux's default congestion control, which runs on top of an unmodified web server (Web1). In the meantime, we also updated Web2 to Linux kernel 4.10 and configured its TCP stack to be BBR. Thus, three different

Table 1. Servers information in the experiment

Machine	Location	CPU	Memory	Bandwidth	OS version
Web1	Guangzhou	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz	4GB	1Gbps	Ubuntu 16.04 + Apache 2.0
Web2	Guangzhou	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz	4GB	1Gbps	Ubuntu 16.04 + Apache 2.0
TCon	Guangzhou	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz	4GB	1Gbps	Ubuntu 16.04 + OVS2.6.0
LAN	Guangzhou	Intel i7-4790 @ 3.6GHz	16GB	1Gbps	Ubuntu 16.04
WAN-Shanghai	Shanghai	Intel(R) Xeon(R) CPU E5-2699A v4 @ 2.40GHz	1GB	5Mbps	Ubuntu 16.04
WAN-Oregon	Oregon	Intel(R) Xeon(R) CPU E5-2686 v3 @ 2.30GHz	1GB	5Mbps	Ubuntu 16.04

Table 2. Different type and size of file in Web server

Size	Type
1.7MB	common pdf file (.pdf)
10MB	common mp3 file (.mp3)
101MB	common video file (.mp4)
562MB	Mininet 2.2.2 image on Ubuntu 14.04 LTS - 64 bit(.zip)
630MB	openSUSE-13.2-NET-i586(.iso)

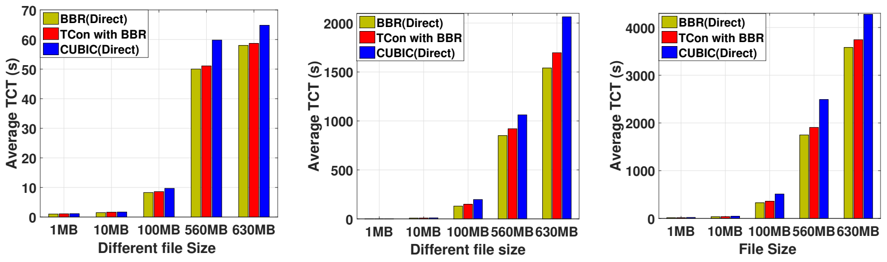
congestion control configurations are considered. (a) *TCon with BBR*: clients connect to Web1 through *TCon*, which enforces BBR to the connection transparently. (b) *CUBIC (Direct)*: clients connect to Web1 directly and the effective congestion control algorithm is CUBIC. (c) *BBR (Direct)*: clients connect to Web2 directly and the effective congestion control algorithm is BBR.

To obtain an in-depth understanding of *TCon*, we designed a variety of benchmarks for performing comprehensive controlled experiments. And these benchmarks involve diverse clients locations and network environments. So, we setup another three servers as clients. One is located in the campus LAN (Guangzhou China). Another two are located in Shanghai China (WAN-Shanghai) and Oregon USA (WAN-Oregon) respectively, connecting to the campus datacenter through the Internet. These clients experienced different RTT and packet drop rate when connecting to web servers. See Table 1 for all servers.

The metrics used are: transfer completion time (measured by CURL) and CPU usage (measured by *top*). We uploaded several files, sized from 1MB to 630MB (see Table 2), to quantify transfer completion time (TCT) for different size files. For each environment, we conducted experiments of all kind of files transferring for about 48 hours. Specifically, we focus on the transfer of 10MB and 630MB files which represent small file and large file respectively.

4.2 Latency Performance

First we evaluated the average transfer completion time (TCT) of different files in different environment. Figure 4 shows the results. Among all machines,



(a) LAN (b) WAN-Shanghai (c) WAN-Oregon

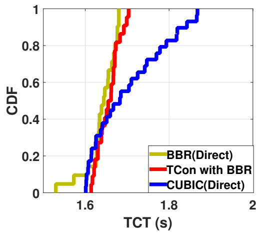
Fig. 4. The average transfer completion time in different environment.

BBR has the best performance while CUBIC is the worst. And *TCon* is better than CUBIC, staying close to BBR.

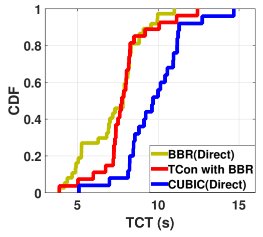
In LAN, the average TCTs for small files of BBR, *TCon* and CUBIC are 1.586s, 1.653s and 1.7003s respectively. Compared to CUBIC, BBR and *TCon* can reduce average TCT by 7.21% and 2.86%. In the meantime, for large files, the average TCTs of BBR, *TCon* and CUBIC are 58.0184s, 58.6933s and 64.8174s respectively. Taking CUBIC as the baseline, BBR and *TCon* can get 11.72% and 10.43% improvement respectively. Then, we have a look at the performance of these three schemes in WAN-Shanghai. For small files, the average TCTs of BBR, *TCon* and CUBIC are 8.3658s, 8.8109s and 10.0857s respectively. Compared to CUBIC, BBR and *TCon* can reduce average TCT by 20.56% and 14.47%. For large files, the average TCTs of BBR, *TCon* and CUBIC are 1540.685s, 1696.7482s and 2062.9752s respectively. Taking CUBIC as the baseline, BBR and *TCon* can get 33.90% and 21.58% improvement. Last, we evaluate the performance in WAN-Oregon. For small file, the average TCTs of BBR, *TCon* and CUBIC is 35.722s, 38.831s and 43.519s respectively. For large files, the average TCT of *TCon* and CUBIC are 3581.961s, 3744.717s and 4276.137s respectively. *TCon* reduces TCT by 12.07% and 14.19% for small files and large files when compared to CUBIC while BBR reduces TCT by 21.83% and 19.38%.

Figure 5 shows the overall performance CDF for transferring small files. Specially, BBR and *TCon* can reduce 99.9^{th} percentile TCT by 11.24% and 9.69% when compared to CUBIC in LAN. Most transfers can finish their transfers within 1.67s,1.7s and 1.85s for BBR, *TCon* and CUBIC respectively. In WAN-Shanghai, BBR and *TCon* reduce 99.9^{th} percentile TCT by 34.89% and 19.04% respectively compared to CUBIC. Most transfers can complete their transfers within 10s, 11s and 14s for BBR, *TCon* and CUBIC respectively. In WAN-Oregon, most file can finish transfer within 41s, 44s and 46s under BBR, *TCon* and CUBIC respectively. Moreover, *TCon* reduces 99.9^{th} percentile TCT by 4.77% compared to CUBIC while BBR reduces 8.48%. We found that the TCT of CUBIC is less than BBR and *TCon* in some cases for LAN. This is because that client of LAN is close to the Web1 and few packets drop and retransmission occur (CUBIC is a loss based).

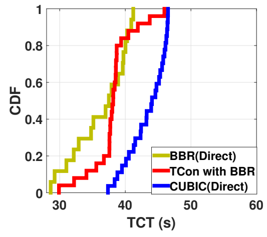
For large file, Figure 6 shows the CDF of TCT in various environment. In LAN environment, BBR and *TCon* reduce 99.9^{th} percentile TCT by 16.87%



(a) LAN

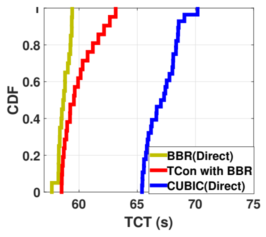


(b) WAN-Shanghai

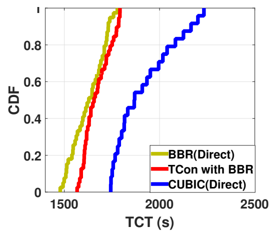


(c) WAN-Oregon

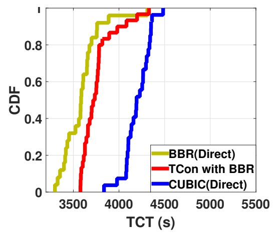
Fig. 5. The CDF of transfer completion time for small file (10MB).



(a) LAN



(b) WAN-Shanghai



(c) WAN-Oregon

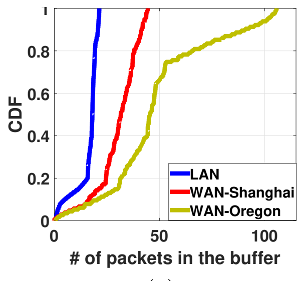
Fig. 6. The CDF of transfer completion time for large file (630MB).

and 10.71% respectively. Most transfers can finish within 59s, 64s and 69s for BBR, *TCon* and CUBIC respectively. In WAN-Shanghai, BBR and *TCon* reduce 99.9^{th} percentile TCT by 25.99% and 24.93% respectively. Most transfers can finish within 1750s, 1780s and 2200s for BBR, *TCon* and CUBIC respectively. Last, in WAN-Oregon, BBR and *TCon* reduce 99.9^{th} percentile TCT by 5.21% and 4.77% compared to CUBIC. And most TCTs are less than 4000s, 4300s and 4400s for BBR, *TCon* and CUBIC respectively.

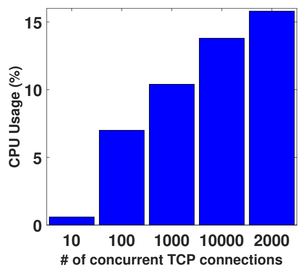
4.3 Overhead of TCon

Also, we have evaluated the overhead of our *TCon*. The buffer size of *TCon* is measured by triggering large file transfers from LAN, WAN-Shanghai and WAN-Oregon. Figure 7(a) depicts the CDF of the number of buffering packets in these three conditions. Results show that, most of time, *TCon* buffers around 20, 40, 105 packets for a LAN, WAN-Shanghai and WAN-Oregon transfers respectively. For a TCP connection, client would respond acknowledge segment as they receive the data from server and generate ACK segment in a short time. So, the number of inflight packets is relatively small.

CPU overhead of *TCon* is measured by simulating concurrent connections. Multiple simultaneous TCP flows are started from LAN server to the Web1 via *TCon* by using Web Bench [2]. Figure 7(b) shows the CPU overhead of *TCon* (the CPU usage of OVS process) and in the worst case with 20000 TCP connections, the maximum CPU usage of *TCon* is about 15%.



(a)



(b)

Fig. 7. *TCon* overhead: (a) CDF of the number of buffering packets and (b) CPU usage of under different scale of concurrent TCP connections.

5 Related Works

The study of TCP congestion control is not new. Many congestion control algorithms have been proposed to reduce latency and improve performance. Reno [8] is nowadays considered the standard TCP which basically implements the four classical congestion control mechanisms of TCP (i.e., Slow Start, Congestion Avoidance, Fast Retransmission and Fast Recovery). Vegas monitors changes in the flow rate (and RTT) to predict congestion before losses occur and intends to reach the expected rate. BIC [14] uses a linear increase to approach a fair window size, and a binary search to improve RTT fairness. While CUBIC [10], which is an improvement of BIC, uses a cubic function to simplify the congestion window computation. BBR, which is proposed in [4], modifies *cwnd* according to the product of propagation time and bottleneck bandwidth.

Rather than proposing a new congestion control algorithm, our work investigates if congestion control can be implemented in a overlay manner. AC/DC [11] and vCC [6] are frontiers which converts default congestion control into operator-defined datacenter TCP congestion control. However, these two schemes target in intra-DC network environment and lack effective approaches to raise sending rate. In WAN, we need a more aggressive mechanism to handle packet loss.

6 Conclusions

Each congestion control mechanism has its own suitable role to play in various network environments. Deploying a specific congestion control algorithms transparently in cloud datacenters is not an easy task. In this paper, we presented *TCon*, a transparent congestion control deploying platform, which aims to enforce more appropriate congestion control algorithm to reduce the WAN transfers latency. Our extensive test-bed results have demonstrated the effectiveness of *TCon* with affordable overhead.

7 Acknowledgements

This work is partially supported by Chinese National Research Fund (NSFC) No. 61402200; NSFC Key Project No. 61532013; NSFC Project No. 61602210; National China 973 Project No. 2015CB352401; the UK Engineering and Physical Sciences Research Council (EPSRC) grants EP/P004407/1 and EP/P004024/1; Shanghai Scientific Innovation Act of STCSM No.15JC1402400 and 985 Project of SJTU with No. WF220103001; the Science and Technology Planning Project of Guangdong Province, China (2014A040401027, 2015A030401043), the Fundamental Research Funds for the Central Universities (21617409, 21617408); the Opening Project of Guangdong Province Key Laboratory of Big Data Analysis and Processing (2017009).

References

1. Open vSwitch, `http://openvswitch.org/`
2. Web Bench 1.5, `http://home.tiscali.cz/~cz210552/webbench.html`
3. Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, J., Gjessing, S., Fairhurst, G., Griwodz, C., Welzl, M.: Reducing internet latency: A survey of techniques and their merits. IEEE Communications Surveys & Tutorials 18(3), 2149–2196 (2014)
4. Cardwell, N., Cheng, Y., Gunn, C.S., Yeganeh, S.H., Jacobson, V.: BBR: congestion-based congestion control. Queue 60(2), 58–66 (2017)
5. Chen, X., Zhai, H., Wang, J., Fang, Y.: A survey on improving tcp performance over wireless networks. Resource management in wireless networking pp. 657–695 (2005)
6. Cronkite-Ratcliff, B., Bergman, A., Vargaftik, S., Ravi, M., Mckeown, N., Abraham, I., Keslassy, I.: Virtualized congestion control. In: ACM SIGCOMM 2016. pp. 230–243 (2016)
7. Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., Govindan, R.: Reducing web latency: the virtue of gentle aggression. In: Acm Sigcomm Conference on Sigcomm. pp. 159–170 (2013)
8. Floyd, S., Gurtov, A., Henderson, T.: The newreno modification to tcp's fast recovery algorithm (2004)
9. Gill, P., Jain, N., Nagappan, N.: Understanding network failures in data centers: measurement, analysis, and implications. In: ACM SIGCOMM Computer Communication Review. vol. 41, pp. 350–361. ACM (2011)
10. Ha, S., Rhee, I., Xu, L.: CUBIC: a new tcp-friendly high-speed tcp variant. Acm Sigops Operating Systems Review 42(5), 64–74 (2008)
11. He, K., Rozner, E., Agarwal, K., Gu, Y.J., Felter, W., Carter, J., Akella, A.: AC/DC TCP: Virtual congestion control enforcement for datacenter networks. In: ACM SIGCOMM 2016. pp. 244–257. ACM (2016)
12. Jacobson, V., Braden, R., Borman, D.: TCP Extensions for High Performance. RFC Editor (1992)
13. Judd, G.: Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In: 12nd USENIX NSDI. pp. 145–157 (2015)
14. Xu, L., Harfoush, K., Rhee, I.: Binary increase congestion control (BIC) for fast long-distance networks. Proc IEEE Infocom Mar 4(4), 2514–2524 vol.4 (2004)