CaaS: Enabling Congestion Control as a Service to Optimize WAN Data Transfer

Jiahua Zhu, Xianliang Jiang(✉), Guang Jin, and Penghui Li

Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China
1811082205@nbu.edu.cn, jiangxianliang@nbu.edu.cn

Abstract. TCP congestion control is essential for improving performance of data transfer. Traditional TCP congestion control algorithm is designed for the wired network with the assumptive goal of attaing higher throughput as possible for QoE. However, Internet today is constantly evolving and many different network architectures (Cellular network, high BDP network, Wi-Fi network, etc.) coexist for data transfer service. Futhermore, the emerging applications (live video, augmented and virtual reality, Internet-of-Things, etc.) present different requirements (low latency, low packet loss rate, low jitter, etc.) for data transfer service. Unfortunately, operating systems (Windows, MacOS, Android, etc.) today still rigidly stick to the single built-in congestion control algorithm (with Cubic for Linux, MacOS, Android and CTCP for Windows) for all connections, no matter if it is ill-suited for current network condition, or if there are better schemes for use. To tackle above issues, we articulate a vision of providing congestion control as a service to enable: (i) timely deployment of novel congestion control algorithms, (ii) dynamical adaption of congestion control algorithm according to the network condition, (iii) and meeting the diversified QoE preference of applications. We design and implement CaaS in Linux, our preliminary experiment shows the feasibility and benefits of CaaS.

Keywords: Network · Transmission Control Protocol · Congestion control

1 Introduction

Today, almost all Internet applications rely on the Transmission Control Protocol (TCP) to deliver data reliably across the network due to its ability to guarantee reliable data delivery across unreliable network. Although it was not part of TCP's primary design, the most vital element of TCP is congestion control (CC), which significantly determines the performance of data transfer.

Traditional CC is mainly designed for the wired network and with the assumptive goal of achieving high throughput as far as possible to attain ideal QoE. Yet, today, Internet is becoming more and more diverse both in the network technologies and the application requirements for data transfer service.

For example, the wide area network (WAN) is being enriched with many different communication networks (satellite network, Cellular network, high BDP network, etc.) and is constantly changing and evolving. Available bandwidth, round trip time (RTT) and packet loss rate can vary over many orders of magnitude among these different network links. Traditional TCP underperform in such situations because it is designed with assumptions that not valid any longer in current networks, such as low latency, no channel packet losses, no dynamic link handover, etc.

Furthermore, the emerging applications exhibit different QoE preference for data transfer service. For example, online video applications usually prefer smooth data transfer to reduce jitter. Web browser requires short flow complete time (FCT) [1] to reduce page load time. Applications in the mobile device may prefer low packet loss rate to extend battery life. The file transfer and data storage applications usually persuit high throughput. Yet, many other delay-sensitive applications such as online games and virtual reality require low latency to provide real-time interaction interaction and will suffer from the throughput-oriented TCP design. Such diversity of requirements for data transfer service further makes current throughput-oriented TCP which built in the OS kernel *"A Jack of all trades, master of none"* protocol.

To deliver high quality of data transfer across such diversity, network researchers have proposed numerous new congestion control algorithms for different types of application requirements and different kinds of network environments. For example, Scalable TCP [2], YeAH [3], BIC [4], and CUBIC [5] are proposed for the networks with high BDP. Westwood [15] and Veno [16] are designed to perform well in the wireless networks. Exll [6] and C2TCP [7] are proposed to achieve low latency in cellular networks. While many of them outperform the legacy TCP in their target scenario, few of them are deployed in the real networks. This is because that today's operating systems just stick to the unified built-in CC scheme (with Cubic for Linux, MacOS, Android and CTCP for Windows) all the time. This presents a barrier to the timely deploy of novel CC schemes and makes advance protocol off limits to users. Furthermore, previous study [11] have revealed that the performance of different CC schemes varies significantly across various network conditions (e.g., high BDP links, satellite links, wireless and lossy links), there is no single CC scheme that can outperform all others in all network conditions. In such case, consistent use of a single built-in CC will undoubtedly degrade TCP performance.

Hence, we ask a question: *Can we reconstruct the endpoint congestion control architecture to enable more flexible and efficient CC deployment?* In this paper, We put a vision of providing congestion control as a service (CaaS) to offer three important benefits that are missing in today's endpoint congestion control architecture: (i) deployment flexibility of novel CC schemes, (ii) adaptive congestion control according to specific network conditions, (iii) and satisfy the diversified QoE preference of different applications.

The main contributions of this paper are as follows. Firstly, we present the design of CaaS, including four important components: Offline Learning, Online

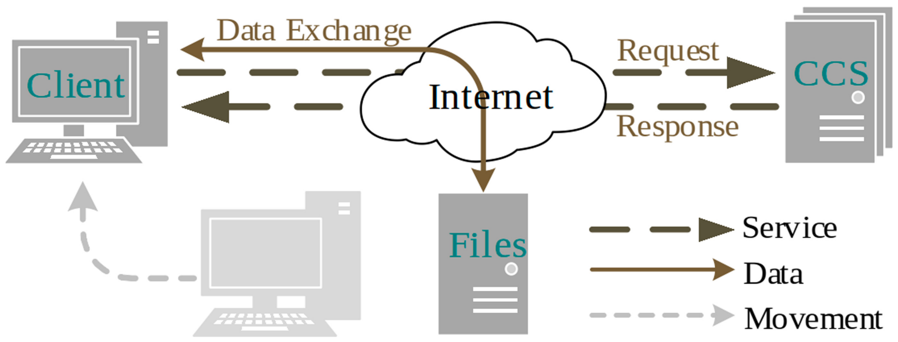

Fig. 1. CaaS architecture.

Matching, Network Change Detection and Algorithm Switching. Secondly, we compare the performance of Caas with legacy congestion control schemes both in the simulation environment and in the real Internet. Preliminary experiment demonstrates the feasibility and superiority of CaaS.

The rest of the paper is organized as follows. Section 2 presents a brief overview of related work. Section 3 details the design of CaaS. In Sect. 4, we evaluate the performance of CaaS. Section 5 provides concluding remarks.

2 Related Work

A. Legacy Congestion Control Schemes. Since the development of Jacobson's TCP Tahoe algorithm in 1988, TCP congestion control over the Internet has been a hot topic for decades because it significantly influences the performance of data transfer. Many researchers have extensively studied and proposed a multitude of enhancements to standard TCP CC. Generally, these CC schemes can be classified into four categories according to their feedbacks: loss-based schemes, delay-based schemes, hybrid schemes and bandwidth-delay-product-based schemes. Loss-based CC protocols, such as HighSpeed [8], BIC, and CUBIC adopt packet loss as the sign of congestion and use an AIMD strategy, which increases the congestion windows (cwnd) if no packet is lost and decreases it on packet losses to avoid copngestion while attaining high network utility. These schemes work well in the wired network with appropriately sized buffers and very little random packet loss. However, they perform poorly in the wireless network which is equipped with large buffers and often experiences high levels of random losses. To tackle with these issues, delay-based schemes, such as Vegas [9] and Hybla [10] use the RTT of packets as an indicator for congestion and keep to reduce queuing delay while achieving high throughput. To take the goodness of both loss-based and delay-based schemes, some hybrid schemes including Compound TCP [12] and TCP Illinois [13] are proposed. They adopt both packet loss and RTT as congestion signal to better predict network congestion. Recently, Google proposed BBR [14], a BDP-based CC protocol which implements a completely different way to implement congestion control. It takes the link bandwidth and the lowest RTT experienced recently to make continuous estimations on bandwidth-delay-product (BDP), and sets the congestion window and pacing rate according to these estimations. It comes out that BBR outperform CUBIC in long fat networks.

Interestingly, each of these variants is only suitable for specific network scenario. For example, Highspeed TCP and Compound TCP are for the high-speed and long-distance networks. TCP Westwood and Veno are proposed to work in wireless networks to enhance throughput. TCP-Peach+ [17] and TCP Hybla are recognized for yielding good performance in satellite networks. Verus [18] and Exll are specifically designed for cellular network scenarios. Despite such a huge set of schemes have been proposed, there is still no "the best TCP" that can always attain best performance across all the possible network conditions. Motivated by this observation, instead of adding just another TCP scheme to such a huge pile of current TCP designs, we focus on providing a platform to dynamically and flexibly deploy these schemes (including the future schemes) in endpoint and enable adaptive congestion control according to specific network scenarios and QoE preference of applications.

B. Learning Based Congestion Control Schemes. Recently, many researchers have paid attention to machine learning (ML) technologies for Internet congestion control and proposed many interesting learning based CC. These schemes usually construct objective function which involves the throughput, latency, packet loss rate and optimize the function by machine learning technologies. Particularly, Remy [19] uses an offline-trained machine learning model to learn congestion control rules which determine the congestion window sizes based on the latest network conditions. Although Remy provides an effective way to generate CC automatically via machine learning, the rules it learned are mined from the offline data of given network condition, thus is not suitable for other network scenarios. Indigo [11] is another method of learning based CC scheme with the data collected from real network. Indigo learns to "imitate" the oracle rule offline. The oracle is constructed with ideal cwnds given by the emulated bottleneck's bandwidth-delay product. Aurora [16] employs deep reinforcement learning technologies to generate a policy that maps observed network statistics to proper sending rate that maximize data transfer utility. Despite the offline learning based schemes can outperform the heuristic algorithms in some scenarios, they may perform badly in network scenarios they have not been trained for.

To tackle above problem, PCC [21] and Vivace [22] adopt an online learning method. They attempt to find proper sending rate to optimize the utility function via a trial-and-error mechanism. Although online learning can adaptively adjust its strategy according to network dynamics, its performance may diminish in some cases as their greedy exploration could be trapped at a local optimum. It should also be noted that online learning usually has a long convergence time, thus is not suitable for short lived flows and complex scenarios.

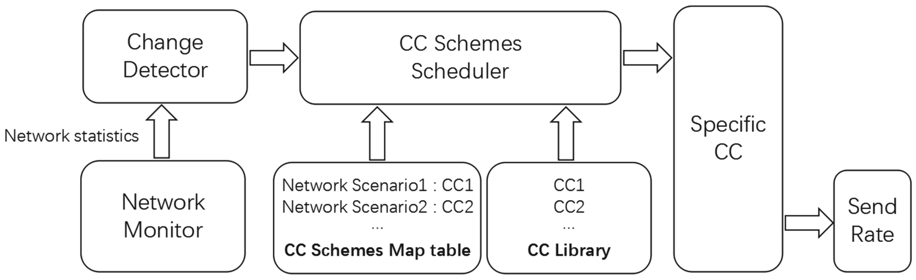

Fig. 2. CaaS Client side framework.

3 System Design

As illustrated above, each CC scheme is only suitable for specific scenario and no single CC scheme is capable of achieving consistent high performance in different network conditions and meet different requirements of applications. Motivated by this, CaaS aims to expedite the deployment of novel CC and distribute the most suitable CC to clients according to network conditions and different QoE requirements of applications in endpoints.

3.1 Overview

At a high-level, CaaS consists of two parts: a server side, termed Congestion Control Server (CCS) and client sides which require congestion control service from CCS, as shown in Fig. 1. The framework of CaaS Client side is shown in Fig. 2. Generally, the server side is responsible for learning the best CC scheme for each network scenario and distributing most suitable CC schemes to clients according to network conditions and application requirements in the client side. The client side can post local network condition information and QoE preference at the suitable time to pull the required CC from CCS, and ship it as a library for congestion control. The downloaded CC modules will be stored in local for future use. We adopt this C/S architecture for timely deployment of new CC schemes.

Congestion control designers can readily publish their new CC scheme by submitting it to the CCS. Each scheme published is required to submit the source code, the design specification for censorship and a statement in the abstract describing scenario where the protocol is recommended or not for use. After that, the CCS will conduct a comprehensive evaluation to check its safety and effectiveness, and decide whether or not to accept it. If adopted, the CC library which holds a set of CC schemes for different application scenarios and the mapping table which maps the network condition to the most suitable CC will be updated to take in new schemes. Some old CC schemes may be replaced if new scheme is more effective in the same or overlapping scenarios. Clients will periodically (about 1–2 weeks) requests the latest mapping table (just 1–3 KB) from the CCS and update the local replica in case that any new CC is adopted in the CCS. We detail main components of CaaS in the following subsections.

3.2 Offline Learning

The optimal CC scheme on a specific network condition can be different across different network conditions, thus it is desirable to find the best CC for every possible network conditions. The offline learning module is responsible for learning the most suitable CC for every given network scenario and output a mapping table which maps a given network condition to the most suitable CC.

To cope with the huge diversity of possible network conditions, we adopt a "divide and conquer" tactic. That is, we divide the possible network conditions in the real world into several sub-scenarios according to the given metric (throughput, rtt, loss rate). In this paper, we assume that the possible network condition varies from 1 Mbit/s to 51 Mbit/s in bandwidth, 10 ms to 200 ms in propagation delay and 0% to 10% in packet loss rate. Then we quantizes each of these three metrics using a quantum (in our experiments, 5 Mbps of bandwidth, 19 ms of delay, 2% of packet loss rate), and obtain 500 sub-scenarios. We simulate each of these sub-scenarios using network simulator Mahimahi [24]. For each sub-scenario, we evaluate the performance of all the CC scheme in the CCS and obtain a performance vector <throughput, delay, loss rate> for each CC scheme. The learning module in the CCS takes the set of performance vectors and determines the best CC for each sub-scenario using a utility function:

$$Utility = w1 * throughput - w2 * delay - w3 * loss, \tag{1}$$

where *w1*, *w2*, *w3* are determined by QoE preference of applications. Note that we provide a QoE Control Panel (QCP) in the client side for users to specify customized QoE requirements for different applications. CaaS client will adopt the default parameters if QoE option is not designated by users. Except for the above utility function, we also provide other QoE evaluation indicator, such as flow complete time (FCT) to satisfy QoE requirements for different applications. Finally, we build the mapping table which points out the best CC for every given network condition. Note that every CaaS client will periodically (about 1–2 weeks) access the CCS to get the latest version of this mapping table for online match of best CC.

3.3 Online Matching

When a TCP connection is established, we firstly adopt the default CC scheme (Cubic), because there is no information about network characteristics to decide optimal CC. Note that every CC schemes is responsible for detect network condition while implementing congestion control. So after a while of data transmission, the sender can preliminarily determine the current network condition and find the most suitable CC scheme according to the mapping table. If the desired CC scheme exists locally, client will switch to it directly. Otherwise, client will

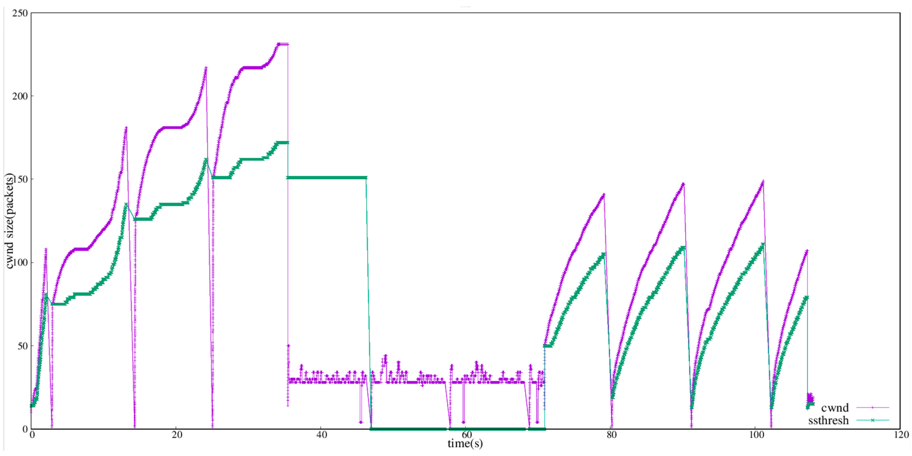

Fig. 3. Hot switch of congestion control schemes.

make a request to the congestion control server (CCS) and switch to the new CC scheme after finishing download. In CaaS, a single CC scheme is approximately 1–3 KB in size, thus it usually takes just a few RTT of time for downloading from server and will not cause too long latency. If the TCP connection is closed when finishing downloade, the new CC will be stored in local for use of next time.

As network condition might change during data transmission, which is common especially for long lived flows, ChangeDetector is continually fed with observations of the network performance metric (throughput, rtt, loss rate) and detect if state has changed. If so, an algorithm switching will be executed to adapt new network condition. We detail Change point detection in next subsection.

3.4 Change Point Detection

Prior work has shown that the network condition along a TCP session is not necessarily a stationary process and might change at different times. For example, mobile phone users in the high-speed rail will undergo frequent handoffs between cellular base stations. Thus, it is desirable to detect such change and adaptively adopt the most suitable CC for new network condition during a TCP connection.

A straightforward way to detect change point of network condition is to continually calculate an exponentially weighted moving average (EWMA) of the related metrics (throughput, RTT, loss rate) samples of TCP flows and check if they reach the threshold values. If so, a new network condition is detected, thus we switch to the new CC for that specific network condition. However, we find this method always leads to frequent and unnecessary switching when these metrics fluctuates, which is very common in practical data transmission.

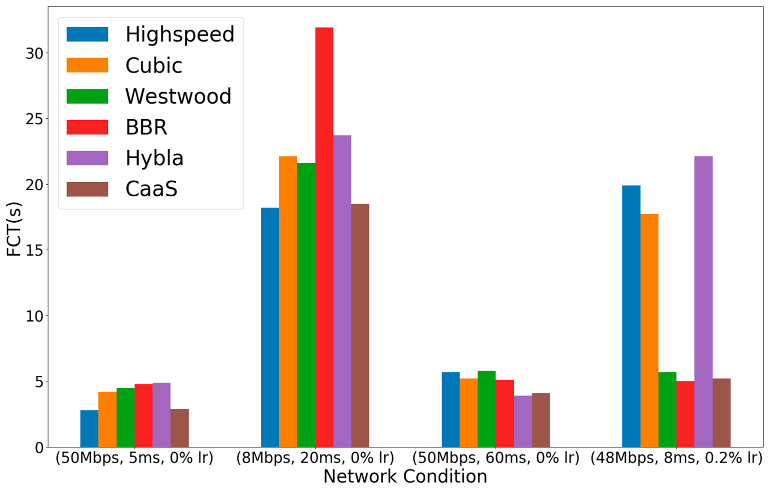

Fig. 4. Flow complete time of different schemes in different network conditions.

Fortunately, according to prior work [25–27], TCP connection metrics can be modeled as a piecewise stationary process which often lasts for tens of seconds or minutes. Motivated by this observation, we use a Bayesian online probabilistic change-point detector [28] which just produce light overhead for computation to detect network condition change. When a TCP connection is created, ChangeDetector is periodically fed with performance metrics, then it calculates mean value and standard deviation of the metrics and detects if new network condition is coming.

3.5 Algorithm Switching

As CaaS dynamically assigns the most suitable congestion control according to connection conditions, algorithm switching should be executed for a single TCP connection when the network condition change is detected. A convenient way to do this is to directly change the congestion control function pointer to another CC module. In this way, however, network throughput will decreases sharply because the sending rate will be initialized to a very small value in the initial phase of a new CC implementation. To ensure smooth transition between algorithms, the new CC is expected to inherite the previous algorithm's sending rate to avoid drastic performance degradation. As for the algorithms employing the pacing mechanism, such as BBR, we set the initial pacing rate to the value of congestion window divided by recent sampled RTT. To test if CaaS client is capable of smooothly switching CC schemes, we switch CC schemes for every 35 s in the order of CUBIC, BBR, Westwood and use tcpprobe tool [23] to observe cwnd. As shown in Fig. 3, each TCP variants perform a different behavior of adjusting cwnd and CaaS is capable of switching different CC at different time.

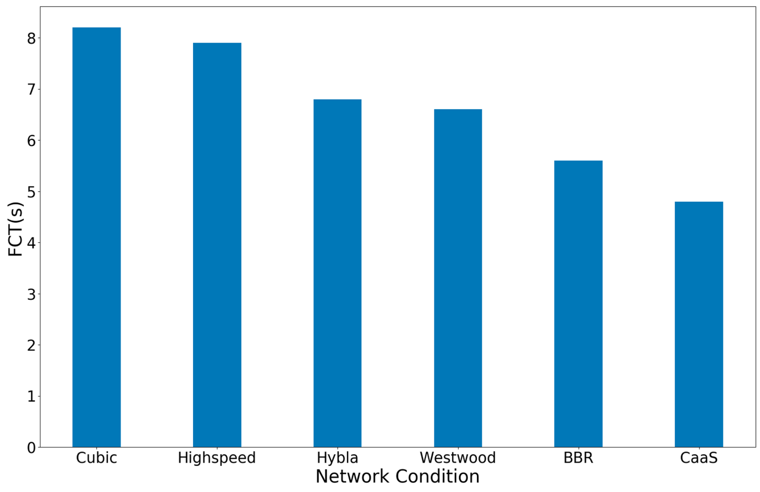

Fig. 5. Flow complete time of different schemes in changing network conditions.

4 Performance Evaluation

To understand and quantify the benefits of CaaS, we evaluated our system both in the network simulator and in the real Internet. Together, these two approaches help us to understand the behavior of CaaS and superiority of it over traditional method. Our test bed consists of 6 clients as data senders and a server as data receiver. In each client, CaaS, Highspeed, Cubic, Westwood, BBR and Hybla are deployed as congestion control scheme respectively. For all clients, we uploaded a file (30 MB) to the server and use transfer completion time (TCT) as the primary performance metric to compare the performance of different CC schemes.

Firstly, we compare the performance of Caas with other schemes in different network scenarios. We randomly generate 4 different network conditions using network simulator Mahimahi. Their link characteristics are (50 Mbps, 60 ms, 0%lr), (50 Mbps, 5 ms, 0%lr), (8 Mbps, 20 ms, 0%lr) and (48 Mbps, 8 ms, 0.2%lr) respectively. In each network condition, clients with different CC schemes send a file to the server. As shown in Fig. 4, every CC scheme has its most suitable network condition and no single CC scheme can outperform all the others in every network condition. Specifically, Hybla achieves best performance in network condition with 50 Mbps bandwidth, 60 ms RTT and 0% loss ratio, but perform worst in other three scenarios. Highspeed outperforms other CC schemes in the first two network conditions but underperforms in the last two scenarios. BBR obtains shortest flow complete time in the network condition with 48 Mbps bandwidth, 8 ms RTT and 0.2% loss ratio, but its performance is significantly worse than other schemes in the network condition with 8 Mbps bandwidth, 20 ms RTT and 0% loss ratio. Fortunately, CaaS always achieves near optimal performance across different network scenarios. This is CaaS can always select the most suitable CC for hosts in different network scenarios.

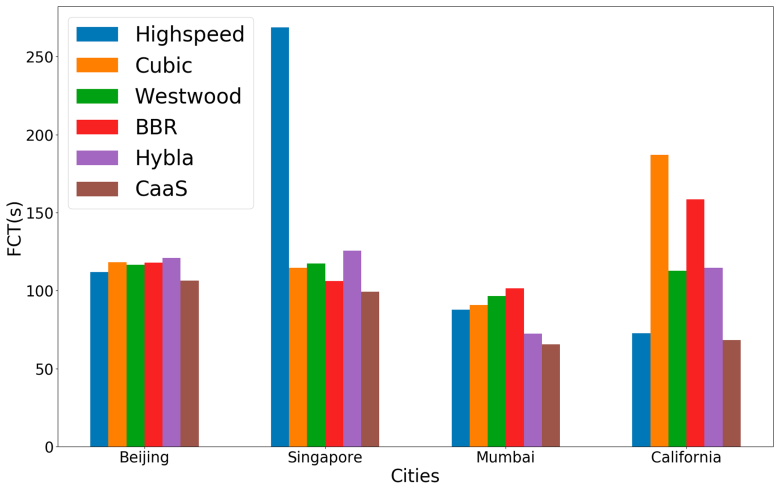

Fig. 6. Flow complete time of different schemes in real Internet.

Secondly, we analyze the performance of CaaS in changing network conditions. The network condition was set to be changed about every 3 s in the order of (48 m, 8 ms, 0.2%lr), (50 m, 60 ms, 0%lr) and (50 m, 5 ms, 0%lr). As shown in Fig. 5, CaaS achieves the shortest flow complete time in data transfer due to its dynamic adjustment CC schemes according to the condition of network. Specifically, we observe that client with CaaS use Cubic as its initial CC, then it switches to BBR at about 0.3 s. After it detects the change of network condition, it switches from BBR to Hybla at about 3.2 s. This proves that CaaS is capable of dynamically switching to the most suitable CC when the network condition is changing.

Thridly, we further evaluate the performance of CaaS in the real Internet. We deploy CaaS in 4 aliyun Web servers from Beijing, Singapore, Mumbai and California respectively. So our local client located in Hangzhou will experience different network characteristics when connecting these Web servers. For each server from different cities, we set client to send a file (300 MB) to server with different CC schemes. As shown in Fig. 6, any specific congestion control algorithm, even the state of the art, cannot excel in diverse network conditions. CaaS can always achieves shortest flow complete time for its adaptive congestion control tactic.

5 Conclusions and Future Work

In this paper, we advocated a vision of providing congestion control as a service to offer flexibility and efficiency benefits for the deploy of novel advanced CC schemes. We implemented a prototype of CaaS to support our argument. Preliminary evaluations confirm that CaaS can transparently deploy suitable CC schemes for clients in different networks, and achieve considerable performance gains as compared to legacy method.

Our prototype system has yielded useful insights for future research of more effective congestion control deploy platform. Our next steps include: 1) further analyze and classify the existing CC schemes to their most suitable network scenario, 2) comprehensively evaluate the overheads of CaaS in terms of CPU usage and extra traffic caused by CC schemes update and 3) improve the performance of the system for short-flow scenarios.

References

1. Dukkipati, N., Mckeown, N.: Why flow-completion time is the right metric for congestion control. ACM SIGCOMM Comput. Commun. Rev. **36**(1), 59–62 (2006)
2. Kelly, T.: Scalable TCP: improving performance in highspeed wide area networks. ACM SIGCOMM Comput. Commun. Rev. **33**(2), 83–91 (2003)
3. Baiocchi, A., Castellani, A.P., Vacirca, F.: YeAH-TCP: yet another highspeed TCP. In: Proceedings of PFLDnet, Roma, Italy, pp. 37–42 (2007)
4. Xu, L., Harfoush, K., Rhee, I.: Binary increase congestion control (BIC) for fast long-distance networks. In: INFOCOM 2004. Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE (2004)
5. Ha, S., Rhee, I., Xu, L.: CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008)
6. Park, S., et al.: ExLL: an extremely low-latency congestion control for mobile cellular networks. In: The 14th International Conference (2018)
7. Abbasloo, S., Li, T., Xu, Y., et al.: Cellular Controlled Delay TCP (C2TCP). arXiv, Networking and Internet Architecture (2018)
8. Floyd, S.: RFC 3649. https://www.ietf.org/rfc/rfc3649.txt. Accessed 10 June 2019
9. Brakmo, L.S., Peterson, L.L.: TCP Vegas: end to end congestion avoidance on a global Internet. IEEE J. Sel. Areas Commun. **13**(8), 1465–1480 (1995)
10. Caini, C., Firrincieli, R.: TCP Hybla: a TCP enhancement for heterogeneous networks. Int. J. Satell. Commun. Netw. **22**(6), 547–566 (2004)
11. Yan, F.Y., Ma, J., Hill, G.D., et al.: Pantheon: the training ground for internet congestion-control research. In: Usenix Annual Technical Conference, pp. 731–743 (2018)
12. Tan, K., et al.: A compound TCP approach for high-speed and long distance networks. In: Infocom IEEE International Conference on Computer Communications. IEEE (2007)
13. Liu, S., Basar, T., Srikant, R.: TCP-Illinois: a loss-and delay-based congestion control algorithm for high-speed networks. Perform. Eval. **65**(6), 417–440 (2008)
14. Cardwell, N., Cheng, Y., et al.: BBR: congestion-based congestion control. ACM Queue **14**(5), 20–53 (2016)
15. Mascolo, S., Casetti, C., et al.: TCP Westwood: bandwidth estimation for enhanced transport over wireless links. In: 7th ACM Conference on Mobile Computing and Networking (MobiCom), Rome, Italy, pp. 287–297 (2001)
16. Fu, C.P., Liew, S.C.: TCP Veno: TCP enhancement for transmission over wireless access networks. IEEE J. Sel. Area. Commun. **21**(2), 216–228 (2003)
17. Akyildiz, I.F., Zhang, X., et al.: TCP-Peach+: enhancement of TCP-Peach for satellite IP networks. IEEE Commun. Lett. **6**(7), 303–305 (2002)

18. Zaki, Y., Poetsch, T., et al.: Adaptive congestion control for unpredictable cellular networks. ACM SIGCOMM Comput. Commun. Rev. **45**(4), 509–522 (2015)
19. Winstein, K., Balakrishnan, H.: TCP ex Machina: computer-generated congestion control. Comput. Commun. Rev. **43**(4), 123–134 (2013)
20. Jay, N., Rotman, N.H., Godfrey, B., et al.: A deep reinforcement learning perspective on internet congestion control. In: International Conference on Machine Learning, pp. 3050–3059 (2019)
21. Dong, M., Li, Q., et al.: PCC: re-architecting congestion control for consistent high performance. In: Networked Systems Design and Implementation, pp. 395–408 (2015)
22. Dong, M., Meng, T., Zarchy, D., et al.: PCC Vivace: online-learning congestion control. In: Networked Systems Design and Implementation, pp. 343–356 (2018)
23. Linux TCP probe. https://wiki.linuxfoundation.org/networking/tcpprobe. Accessed 12 Oct 2019
24. Netravali, R., Sivaraman, A., Das, S., et al.: Mahimahi: accurate record-and-replay for HTTP. In: Usenix Annual Technical Conference, pp. 417–429 (2015)
25. Balakrishnan, H., Stemm, M., et al.: Analyzing stability in wide-area network performance. Meas. Model. Comput. Syst. **25**(1), 2–12 (1997)
26. Jobin, J., Faloutsos, M., et al.: Understanding the effects of hotspots in wireless cellular networks. In: Proceedings of the Conference of the IEEE Computer and Communications Societies, INFOCOM (2004)
27. Lu, D., Qiao, Y., Dinda, P.A., et al.: Characterizing and predicting TCP throughput on the wide area network. In: IEEE International Conference on Distributed Computing Systems, ICDCS (2005)
28. Ryan Prescott Adams and David JC MacKay: Bayesian Online Changepoint Detection. In arXiv:0710.3742v1 (2007)