IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

IJARSCT
Impact Factor: 6.252

Volume 2, Issue 1, August 2022

Slow Start Concept in Data Communication

Likhita KM[1], Mr. Pradeep Nayak[2], Finny Paul[3], Keerthana G[4], Gary Richards[5]

Students, Department of Information Science and Engineering[1,3,4,5]

Assistant Professor, Department of Computer Science and Engineering[2]

Alvas Institute of Engineering and Technology, Mijar, Karnataka, India

4al20is021@gmail.com, pradeep@aiet.org.in, 4al20is018@gmail.com

keerthanasonu007@gmail.com, 4al20is019@gmail.com

Abstract: *The Transmission Control Protocol(TCP) is one of the pillars of the Internet. As a result, a lot of research has been done to enhance its performance, primarily by improving the congestion control algorithm of TCP. In this work, we demonstrate that TCP's Slow-Start algorithm is increasingly turning into a bottleneck in contemporary high-speed networks, in addition to congestion control. By incorporating StopEG into Google's BBR congestion control algorithm, it is evaluated through simulations in ns-3. Results from simulations show that it is effective in BBR, with a 68% reduction in the bottleneck queue's length when new connections are made. We suggest a different method called Stateful-TCP to address the issue, in which the path bandwidth estimated in a prior flow is used to immediately increase the transmission rate of a subsequent flow to the same destination. As a result, conventional Slow-Start is no longer necessary, and TCP is able to utilise the available path bandwidth effectively right away.*

Keywords: Data Communication.

I. INTRODUCTION

Since it is used by the majority of applications to transport data, the Transmission Control Protocol (TCP) is one of the Internet's pillars. Due to this, numerous studies have been conducted to enhance its performance in a variety of network environments (e.g., [1]– [20]). A TCP flow begins in a Slow-Start phase [21] with a relatively low initial transmission rate, which is gradually increased as packets are correctly delivered. This Slow-Start phase is common to almost all TCP variants. Today's networks have a wide range of bandwidth (from Kbps to Gbps, for example), so transmitting too aggressively in the beginning could cause severe congestion in networks with limited bandwidth. However, TCP's Slow-Start mechanism could seriously hinder the protocol's performance, especially in networks with high bandwidth-delay products (BDP). This restriction is well-known, and in the current Linux kernel [1], the initial CWnd of TCP has been increased from 2 MSS in TCP Reno to 10 MSS. Although increasing the initial CWnd can partially alleviate the limitation, many Internet flows are still affected.Themajority of Internet applications start several subsequent TCP flows within a single application session, in contrast to TCP's original design, where each TCP flow is independent. Since the network conditions that these TCP flows are likely to encounter are very similar, much can be inferred from a prior TCP flow to the same peer host. This prompts the development of a novel Stateful-TCP mechanism in which the path bandwidth of a previous flow is estimated to instantly increase the transmission rate of a subsequent flow to the same destination. This completely eliminates the need for Slow-Start, allowing TCP to utilise the available bandwidth effectively from the start.

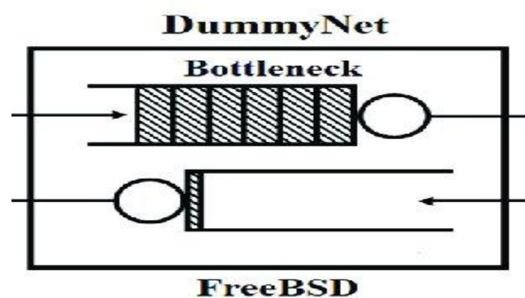
II. BACKGROUND AND RELATED WORKS

Since its introduction, the Transmission Control Protocol has undergone significant development. For instance, Linux Kernel 4.20 implements a total of 17 TCP variants created for various network environments, with Cubic [2] serving as the current default and displacing TCP Reno [22] as a result of Cubic's superior performance over large-BDP networks [2]. Many new TCP variants have been developed over the years. For instance, Westwood [3], Veno [4], and BBR [5] were created to lessen the effects of random loss in mobile networks; TCPHybla [6] was created for a satellite network with a high loss rate and a long delay; DCTCP [7] was created for datacenter networks; Sprout [8] was created for delay-sensitive applications in mobile networks; PCC [9] was created to change its congestion control behaviour based on in-band network measurements; and Most earlier works concentrated on the TCP congestion control algorithm.

Nevertheless, TCP almost always starts a new flow with the conservative Slow-Start to gradually explore the path bandwidth, regardless of the congestion control algorithm. This is essential because the path bandwidth is unknown, but it can also become a significant bottleneck, as was previously discussed. This encourages scientists to create new techniques to deal with the Slow-Start bottleneck.

For instance, Quick-Start was proposed by Hauger et al. [11], which modifies TCP's sending rate based on explicit input from network routers. JumpStart was the method suggested by Liu et al. [12], in which the initial CWnd was set to the receiver's advertised window size and applied pacing to control the initial transmission rate. Jump-Start was further expanded by Li et al. [13] to incorporate proactive retransmission to lower FCT in the event of packet losses near the end of a flow. Using proactive learning at the server-side to dynamically assign the initial CWnd for each new flow based on historical data, Nie et al. [14] proposed TCP-WISE in a more recent work. Winstein and Balakrishnan [15] suggested Remy in a different manner, doing away with the Slow-Start phase entirely and using network knowledge to optimise the congestion-control algorithm. Although Slow-Start starts with a small CWnd, it quickly grows, and in some network configurations, it may also exceed the link capacity, resulting in unneeded congestion. Researchers suggested additional conditions to change the Slow-Start exit point in order to solve this issue. In Floyd's Limited Slow-Start proposal, for instance, a new parameter named max ssthresh was added (default to 100). The CWnd will only increase by a maximum of max ssthresh/2perRTT once it exceeds max ssthresh but falls short of ssthresh. Cavendishetal. [24] proposedCapStart to consider the network interface card's (NIC) bandwidth and only activate Limited Slow-Start if the NIC bandwidth is greater than the estimated path bandwidth. Since then, the Linux kernel has incorporated Hystart and has made it default-enabled. When it triggers, Hystart++ updates CWnd in accordance with Limited Slow-Start until packet loss occurs or the ssthresh is exceeded instead of entering the congestion avoidance phase. Contrarily, on the one hand, Slow-Start frequently becomes a bottleneck because of the low initial CWnd. On the other hand, Slow-Start may also result in congestion if CWnd expands too quickly. As a result, the issue with TCP's startup phase is still far from being resolved. It is obvious that the issue is probably not even amenable to solution in the absence of knowledge of the network path. We claim that TCP's startup phase needs to be further optimised after the initial CWnd optimization. For instance, even with an RTT of 20 ms, at 1 Gbps bandwidth, the CWnd necessary to fully utilise bandwidth will be 2.5 MB in 5G networks that promise bandwidth in excess of Gbps. If left unchecked, a large initial CWnd could result in significant congestion losses.

III. TCP SLOW-START REVISITE

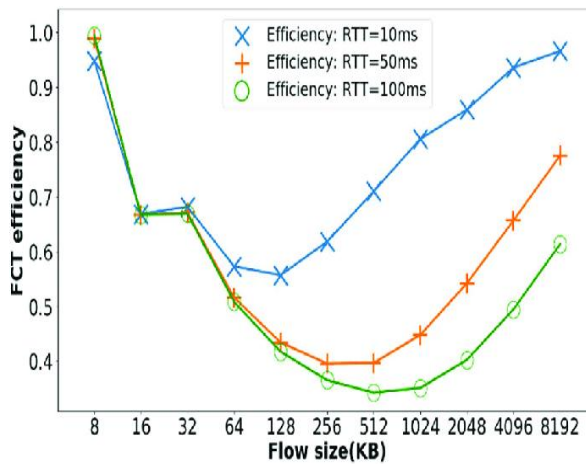


IV. PERFORMANCE IMPACT

In Slow-Start, TCP starts with a modest initial CWnd (for example, 10 MSS in the current Linux kernel) and exponentially increases it as ACKs are received, doubling it every RTT. When the CWnd exceeds the slow-start threshold (ssthresh), Hystart [16] (if enabled) triggers, or a loss event occurs [21], the slow-start phase ends. We conducted experiments using the topology shown in Figure 1 to illustrate the performance impact of the TCPS slow start. Our server and client operating systems were Ubuntu with Linux Kernel 5.3, and we used DummyNet [26] to simulate a bottleneck link with 100 Mbps of bandwidth and RTTo of 10 ms, 50 ms, and 100 ms. There is no random loss, and one BDP was set as the link buffer size. The flow size spans the majority of the flow size distribution on the Internet, from 8 KB to 8.2 MB [20], [28]. For performance comparison, we first calculate the flow completion time (FCT), denoted by T, which is defined as the amount of time from when the client (receiver) sends the TCP SYN segment to when all data are received. We then compare it to the minimum FCT, denoted by Tmin, which is defined as

Tmin =1.5dmin+ S C + 0.5dmin (1), where dmin is the two-way propagation delay (assumed to be The three terms on the RHS stand for the following: I three-way handshaking time; (ii) transmission time; and (iii) one-way propagation delay for the final TCP data segment to arrive at the receiver. Tmin, then, stands for the lowest FCT that any TCP protocol can achieve along the specified network path. We then use it to define the FCT efficiency metric, denoted by, = Tmin T, (2), where it has a range of 0 to 1, with 1 denoting ideal performance. When compared to throughput and utilisation, FCT's ability to account for propagation delay and overheads in TCP's three-way handshake is a desirable feature.

In Slow-Start, TCP starts with a modest initial CWnd (for example, 10 MSS in the current Linux kernel) and exponentially increases it as ACKs are received, doubling it every RTT. When the CWnd exceeds the slow-start threshold (ssthresh), Hystart [16] (if enabled) triggers, or a loss event occurs [21], the slow-start phase ends. We conducted experiments using the topology shown in Figure 1 to illustrate the performance impact of the TCPS slow start Our server and client operating systems were Ubuntu with Linux Kernel 5.3, and we used DummyNet [26] to simulate a bottleneck link with 100 Mbps of bandwidth and RTTo of 10 ms, 50 ms, and 100 ms.There is no random loss, and one BDP was set as the link buffer size.



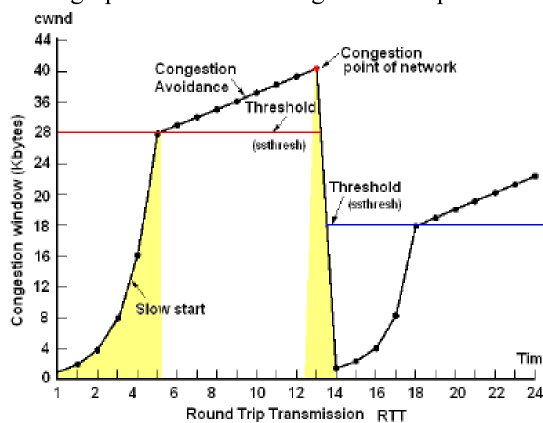
The flow size spans the majority of the flow size distribution on the Internet, from 8 KB to 8.2 MB [20], [28]. For performance comparison, we first calculate the flow completion time (FCT), denoted by T, which is defined as the amount of time from when the client (receiver) sends the TCP SYN segment to when all data are received. We then compare it to the minimum FCT, denoted by Tmin, which is defined as Tmin =1.5dmin+ S C + 0.5dmin (1), where dmin is the two-way propagation delay (assumed to be The three terms on the RHS stand for the following: I three-way handshaking time; (ii) transmission time; and (iii) one-way propagation delay for the final TCP data segment to arrive at the receiver. Tmin, then, stands for the lowest FCT that any TCP protocol can achieve along the specified network path. We then use it to define the FCT efficiency metric, denoted by, = Tmin T, (2), where it has a range of 0 to 1, with 1 denoting ideal performance. When compared to throughput and utilisation, FCT's ability to account for propagation delay and overheads in TCP's three-way handshake is a desirable feature.

more precisely gauges how the user or application feels about the performance. For three different RTT settings, the FCT efficiency of Cubic is plotted against flow size in Figure 2. Ten different experiment runs are repeated to average the results for each setting. We note that the FCT efficiency of Cubic displayed a U-shaped characteristic with respect to flow size. Since Linux TCP has a default initial CWnd of 10 MSS, a flow size no larger than 10 MSS (for example, 8 KB) can be transmitted entirely in the first RTT, achieving an FCT efficiency that is close to ideal. Because host processing delays are not taken into account when calculating the minimum FCT in (1), Cubic did not achieve an exact FCT efficiency of 1 in the experiment. Cubic loses efficiency as flow size grows because its CWnd is smaller than the path's BDP, preventing it from fully utilising the available bandwidth. Its effectiveness gradually increases as flow size further rises as a result of CWnd growth. As would be expected, propagation delay affects Cubic's efficiency, which deteriorates more in the 50 ms and 100 ms cases. In these two instances, Cubic's efficiency falls below 0.5 over a large range of low sizes (e.g.,128KBto1024KB).

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Impact Factor: 6.252
Volume 2, Issue 1, August 2022

V. RELATED WORK

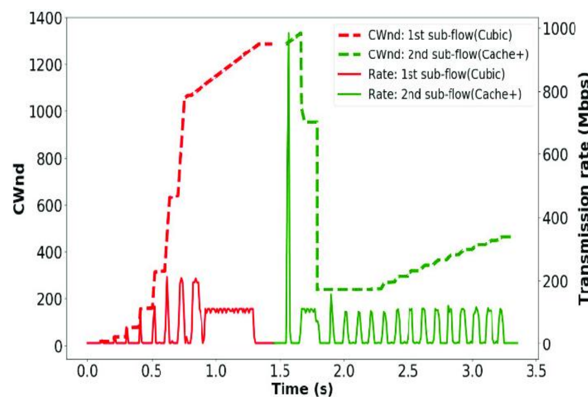
As was previously mentioned, a precise estimate of Kleinrock's optimum point can assist the source in preventing overshooting appropriate cwnd values, further enhancing performance. Kleinrock's point is simple in concept, but it is impossible to determine its precise value in real life [3]. Instead, once packets begin to accumulate at the bottleneck buffer, the existing techniques used in slow-start attempt to reduce the negative effects of cwnd overshooting by abruptly stopping its exponential growth. The time when packets begin to assemble at the bottleneck buffer is indicated in this section by the letter tq. The following techniques are frequently employed in the literature to identify tq: (1) measuring Ack spacing dispersion [11], (2) measuring RTT samples [2], [4], [11], (3) measuring Ack trains [2], [4], and (4) observing a delivery rate plateau [3]. Later in the paper, each of these approaches is covered in more detail. However, it is important to note that RFC 3742 [12] attemptsto prevent cwnd overshooting by setting a strict limit on cwnd size. That is, RFC 3742 switches to limited-slow-start when the limit is reached, regardless of the network conditions. To avoid heavy buffer overflow, limited-slow-start increases cwnd at a comparatively slower rate. Although this approach is straightforward, lightweight, and simple to use, it is not network-dynamics-based [4,] and is not scalable. Additionally, it might put a TCP session in the dangerous region depicted in Fig. 1 where a buffer overflow is likely and the queuing delay is high. When cwnd exceeds BDP and packets experience some queuing delay, CapStart [11] detects a capacity reduction scenario and switches from slow-start to limited-slow-start. It attempts to determine tq by using dispersion measurements of packet pairs—two packets that have been sent back-to-back. In particular, CapStart detects queuing delay by measuring the RTT for two consecutive packets and lowers the sending rate. CapStart has not been widely used in practise because it requires precise time measurement and views any delay greater than the minimum RTT as a likely signal to halt exponential growth. An Ack train is described in HyStart (Hybrid Start) [4] as a line of consecutive Acks that are only a few milliseconds (e.g., 2ms) apart when they are received. In accordance with this definition, HyStart passively tracks both Ack trains and RTT samples to identify a slowstart's safe exit point. Since Linux 2.6.29, CUBIC's HyStart has been included as a modified slow-start algorithm, and TCPNile1—the new CCA from Citrix Systems—also uses it as an option to boost throughput in big BDP networks. HyStart's main goal is to initiate slow-start exit whenever there is an increase in round-trip delay or whenever it takes longer than the minimum RTT to receive an Ack train. In order to avoid being tricked by transient queuing events, the first heuristic only looks at the RTT samples of the first few acks in each ack train (for example, the first eight acks). However, an additional brief burst that has just entered the bottleneck buffer can increase these RTT samples and result in erroneous tq detection. According to the second heuristic, an Ack train is no longer regarded as a train if there is a gap of more than 2ms between two consecutive Acks. This typically occurs when multiple connections are utilising the same bottleneck link. Consequently, tq detection is put off until the next train that arrives in the following round, where the same issue might still exist. HyStart++ [2], which combines RFC 3742 with HyStart, was recently proposed with the goal of preventing burst losses by lowering the sending rate while still maintaining high throughput. It starts slowly and doubles cwnd every RTT until the delay increases, much like HyStart. Then it changes to limited-slow-start, where cwnd is cautiously increased to delay any potential buffer overflow and lessen its negative effects. HyStart++, which is integrated into the TCP stack of Microsoft Windows, has been widely used online. Although it addresses the scalability problem in RFC 3742,its approach to finding tq is based on looking for RTT spikes which are not always reliable signs.



VI. THE CWND DILEMMA

to impose a strict limit on cwnd size in order to prevent cwnd overshooting. In other words, regardless of the state of the network, RFC 3742 switches to limited-slow-start when the limit is reached. Limited-slow-start raises cwnd at a comparatively slower rate to prevent heavy buffer overflow. This method is straightforward, light-weight, and easy to use, but it is not scalable or based on network dynamics [4,]. Additionally, it might place a TCP session in the risky area shown in Fig. 1, where a buffer overflow is probably going to happen and the queuing delay is going to be significant. CapStart [11] recognises a capacity reduction scenario when cwnd exceeds BDP and packets experience some queuing delay and switches from slow-start to limited-slow-start. It makes an effort to determine tq using measurements of the dispersion of packet pairs—two packets thatBy measuring the RTT for two consecutive packets, CapStart, in particular, can identify queuing delays and reduces sending ratesincrease in throughput for large BDP networks.The bottleneck link is being used by many connections.

VII. STATEFUL-TCP



7.1 Principle

he main problem with Slow-Start is that a CWnd that is too small will result in under-utilization of the available bandwidth. A large enough CWnd, though, might also cause a buffer overflow at the bottleneck link. A crucial realisation for resolving this conundrum is that the initial burst transmission, not the large CWnd, is what's causing the buffer overflow. As a contrast, if one runs an extended TCP flow, its CWnd can also increase to a high value, but this is not always indicative of buffer overflow. This is so because, at steady state, the bottleneck link, which also serves as the clock for the receiver's ACKs, also serves as the clock for TCP's packet transmissions. Transmissions are therefore timed to match the bottleneck link bandwidth. For TCP-Cache and TCP-Cache+, on the other hand, there is no such pacing at the start, resulting in a significant transmission burst. Stateful TCP's objective is to enable new TCP flows to operate in the same manner as if they are continuations of previous flows to the same destinations, bypassing slow starts and transmitting at the path bandwidth from the outset. The system design for achieving this goal is presented in the following section.

7.2 System Design

Stateful-TCP operates in three phases: Startup phase, Estimation phase, and Termination phase, as depicted in Fig. 5.A hash table is used to cache states from a completed flow. Each table entry comprises three fields: peer IP address, estimated bandwidth, and minimum RTT.

7.3 Startup Phase

Miss - The regular Slow-Start is used because the hash table entry is empty, i.e., there is no record of a previous flow. Stateful-TCP will be activated for the new flow by: (a) setting CWnd to the previous flow's BDP, i.e., estimated bandwidth minimum RTT; (b) suppressing AWnd (except when 0); and (c) activating pacing for outgoing packets at a rate equal to the previous flow's estimated bandwidth. Hit - The hash table entry is non-empty and the previous flow's IP address matches the new one. Collision - The hash table entry is not empty, but the IP address of the previous flow

ISSN (Online) 2581-9429
IJARSCT
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Impact Factor: 6.252
Volume 2, Issue 1, August 2022

and the new one are different, indicating a hash collision. In this situation, the new flow will resume its regular Slow-Start. States of the new TCP flow will be stored to the table entry when the flow ends in both miss and collision cases for the hash table. One exception is when the estimated BDP is equal to or less than TCP's default initial CWnd, indicated by CWndinit (10MSS in Linux), in which case the states will be ignored because there is no performance advantage over regular Slow-Start. Pacing will be turned off and CWnd will be set to CWnd = max, CWndinit, (3) when the first ACK is received, marking the conclusion of the start-up phase, where denotes the number of packets currently in flight. According to the theory, the amount of packets currently in flight corresponds to the path's BDP, which is what CWnd should be in order to utilise the available bandwidth to its fullest. In order to prevent CWnd from being lower than standard Slow-Start, we set a minimum of CWndinit.

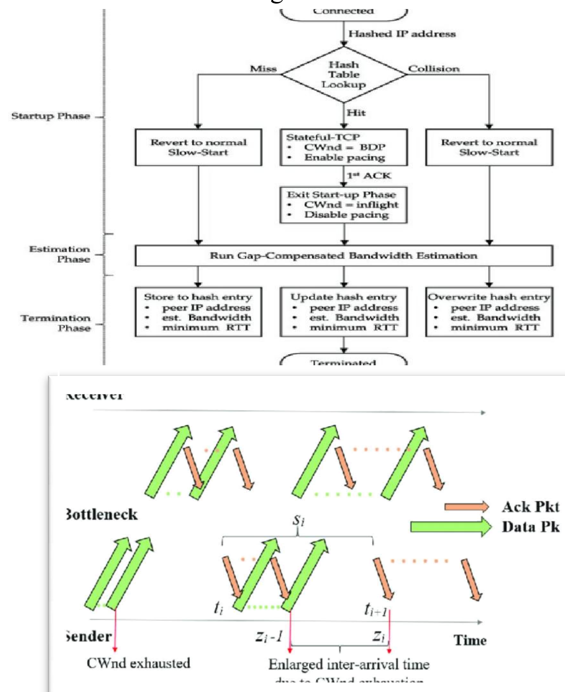
With a bottleneck link bandwidth of 100 Mbps and a link buffer size of one BDP, FIGURE 6 compares bandwidth estimation versus flow size.

Phase 2: Estimation This phase immediately follows the Startup phase and lasts until the TCP flow is stopped. The TCP sender continuously calculates the path bandwidth during this time (c.f. Section IV-C). Because the estimation process runs separately from the congestion control algorithm, only one aspect of the latter's operations must change: the AWnd is suppressed except for 0 to prevent it from becoming the bottleneck (c.f. Section III-C).

Phase 3: Termination Stateful-TCP will add the estimated bandwidth, the minimum RTT, and the client's IP address to the hash table entry upon connection closure, such as when it receives a FIN segment from the client.

VIII. BANDWIDTH ESTIMATION WITH GAP COMPENSATION:

A crucial element of Stateful-TCP is the bandwidth estimator, which operates all through Estimation. Path bandwidth estimation is a well-known issue. In fact, some notable TCP variants, like Westwood [3] and BBR [5], heavily rely on bandwidth estimation. Similar in concept, one should estimate bandwidth in accordance with the volume of data acknowledged within a predetermined amount of time (e.g., onesmoothedRTT). However, these algorithms may be underestimated when used with Stateful-TCP. We first investigate the potential issue by adding instrumentation codes to Westwood and BBR's Linux implementation to record the bandwidth that is estimated upon flow termination. With a bottleneck link bandwidth of 100 Mbps and a bottleneck link buffer size of one BDP, we ran experiments using the topology in Fig. 1. Linux kernel version 5.3 was used by both the client and the server. For flow sizes ranging from 4 KB to 8.2 MB, the bandwidth estimated by Westwood and BBR is plotted in Fig. 6. The first finding is that for smaller flow sizes, both algorithms significantly underestimate the link bandwidth (100 Mbps). For instance, only at the largest links does Westwood's estimated bandwidth start to converge to the actual link bandwidth.



ISSN (Online) 2581-9429

IJARSCT

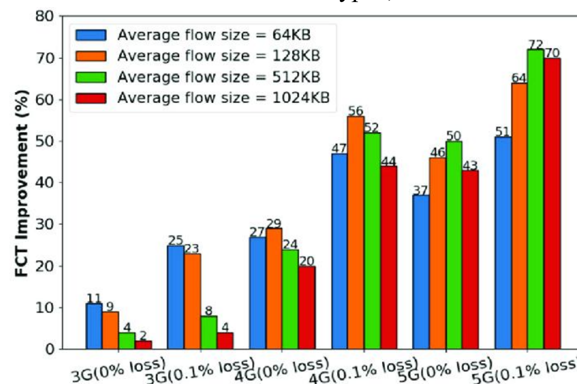
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Volume 2, Issue 1, August 2022

Impact Factor: 6.252

IX. MOBILE NETWORK EXPERIMENTS

Mobile networks may be challenging for S-Cubic because it uses the estimated bandwidth from the previous flow as the initial sending rate. Mobile networks are known to exhibit rapid and significant bandwidth fluctuations over short timescales[18],[19],[44].width data traces. In 3G and 4G networks, the link buffer size was set to 1,280 KB and 5MB, respectively, and the RTT was set to 50 ms for both 3G and 4G. According to our measurement of the actual 5G network, the RTT and link buffer size for 5G were set to 20 ms and 10 MB, respectively. 0% and 0.1%[46] were tested as two random loss rates. S-FCT Cubic's advancements over Cubic in mobile networks are summarised in Fig. 16. There are two things to note. The performance gains for S-Cubic first rise from 3G to 4G and then again in 5G. This is because the BDPs for 4G and 5G are different from those of 3G and are 84 MSS and 305 MSS, respectively. This strongly suggests that TCP may become an increasingly important bottleneck in 5G and other future high-speed mobile networks, making it difficult to fully utilise the bandwidth provided by these networks. As additional proof, a tier-one service provider in China conducted a separate ongoing measurement of a production mobile service and found that the average TCP throughput for a 1 MB data transfer for users connecting over 5G networks is an unanticipatedly low 16 Mbps. Although the physical bandwidth is indeed very high, current TCP cannot fully utilise it, as shown by our UDP-based measurement of a production 5G network in China, which shows an average bandwidth of 180 Mbps. Second, we can see in Fig. 16 that the performance gains for S-Cubic are higher when 0.1% random loss is introduced when comparing the cases with and without random packet loss. This seems counterintuitive because, aside from the Startup phase, S-Cubic operates exactly like regular Cubic. It turns out that the distinction in the Startup phase is precisely the cause of the problem. Random loss is known to specifically interfere with Cubic's Slow-Start phase [3]–[5], which prolongs the time it takes for Cubic to ramp up its CWnd. S-Cubic, in contrast, completely avoids Slow-Start, making it much less susceptible to random packet loss. The delay performance of Cubic and S-Cubic is then compared in Table 5. The most notable variations are seen between various network types, with3 Gexhibiting

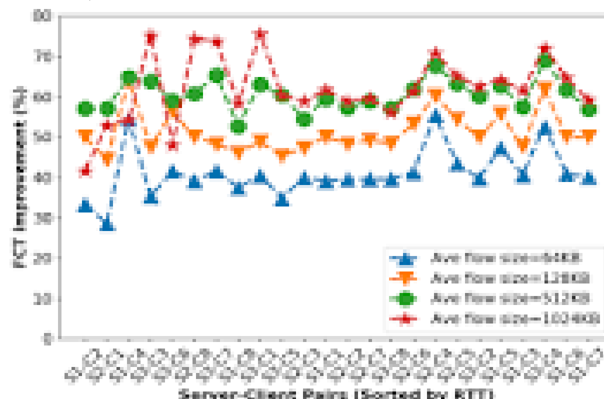


X. INDEPENDENT BENCHMARKS

In this section, we present the findings of an independent performance benchmarking firm from China called Bonree [47]. This company specialises in conducting independent network performance benchmarks, the results of which are primarily used by their clients to assess and compare hosting companies. In the benchmark setup (Table 6), there are more than 1,000 end user clients dispersed across nine Chinese provinces. Wireless or wired networks are used to connect clients. To avoid targeted optimization, detail locations or network access information are not disclosed. This benchmark's scope and geographic coverage provide a realistic glimpse into how Cubic and S-Cubic perform in practical settings. The 8-hour mean FCT over the course of one week is summarised in Fig. 19(a). Our observations show that S-Cubic consistently produced significantly shorter mean FCT than Cubic. The average FCT for Cubic and S-Cubic was 0.40 and 0.25 respectively, an improvement in FCT of 37.5% for S-Cubic. In just a handful of the 10,000 downloads, Cubic had average throughput of more than 100 Mbps, whereas in S-Cubic, 10% of downloads had throughput of more than 100 Mbps. Similar to this, Cubic and's overall average throughput.Experiment Settings Adopted in the Independent Benchmarking platform Bonree [47].

IJARSCT
ISSN (Online) 2581-9429
IJARSCT
Impact Factor: 6.252
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, August 2022

Comparison of average FCT for Cubic, BBR and S-Cubic obtained from Bonree.



XI. SUMMARY AND FUTURE WORK

This work introduces a new way to overcome the limitation of TCP Slow-Start, which is increasingly becoming themodern high-speed networks bottleneck The emulation results and experimental findings from real-world networks demonstrate that applying Stateful-TCP to Cubicles has the potential to produce significant performance gains that may even be necessary to take advantage of future high-speed networks like 5G and beyond. This study is just the beginning. There are still a lot of unresolved issues that demand further research. For instance, applying Stateful-TCP to other TCP variants would be a fruitful area for future research, particularly those created for different purposes, such as lower latency[8],[10], and very large BDP[6]. Stateful-TCP, on the other hand, currently only speeds up the startup process. Congestion control may also be able to benefit from some of the Stateful-TCP techniques, such as GCBE, to further enhance performance after the startup phase. Despite the promising findings of this study, more testing and validation of StatefulTCP in general, and S-Cubic in particular, are required to better understand how well it performs in a variety of environments and how it might affect other traffics that are constrained by the same bottleneck. In order to measure the performance improvements at the application level, the authors are working with Tencent to scale up the Internet experiments and expand them into actual Internet services. Additionally, we are making the Linux S-Cubic implementation available as open-source software1 so that the research community can review it, independently verify its performance, and use it as a model for implementing Stateful-TCP on other TCP designs.

ACKNOWLEDGMENT

The authors would like to thank the associate editor, anonymous reviewers, and Tencent for their suggestions that helped shape this article into its final form and for their generous support of this work by providing access to the Bonree benchmarking platform and Tencent cloud instances for the experiments.

REFERENCES

[1]. N.Dukkipati,T.Refice,Y.Cheng,J.Chu,T.Herbert,A.Agarwal,A.Jain, and N. Sutin, ‘‘An argument for increasing TCP's initial congestion window,’’ACMSIGCOMMComput.Commun.Rev.,vol.40,no.3,pp. 26–33, 2010.

[2]. I. Rhee and L. Xu, ‘‘CUBIC: A new TCP-friendly high-speed TCP variant,’’ ACM SIGOPS Operating Syst. Rev., vol. 42, no. 5, pp. 64–74, 2008.

[3]. S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, ‘‘TCP westwood: Bandwidth estimation for enhanced transport over wireless links,’’ in Proc. 7th Annu. Int. Conf. Mobile Comput.Netw. (MobiCom), Rome, Italy, 2001, pp. 287–297.

[4]. C. P. Fu and S. C. Liew, ‘‘TCP veno: TCP enhancement for transmission over wireless access networks,’’IEEEJ.Sel.AreasCommun.,vol.21,no.2, pp. 216–228, Feb. 2003.

[5]. N. Cardwell, Y. Cheng, C. S. Gunn, V. Jacobson, and S. Yeganeh, ‘‘BBR: Congestion-based congestion control,’’ Queue, vol. 14, no. 5, pp. 20–53, Sep. 2016.

[6]. C. Caini and R. Firrincieli, ''TCP hybla: A TCP enhancement for heterogeneous networks,'' Int. J. Satell. Commun.Netw., vol. 22, no. 5, pp. 547–566, Sep. 2004.

[7]. M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, ''Data center TCP (DCTCP),'' in Proc. ACM SIGCOMM Conf. SIGCOMM, New Delhi, India, Aug. 2010, pp. 63–74.

[8]. K. Winstein, A. Sivaraman, and H. Balakrishnan, ''Stochastic forecasts achieve high throughput and low delay over cellular networks,'' in Proc. NSDI, Lombard, IL, USA, Apr. 2013, pp. 459–471.

[9]. M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, ''PCC: Rearchitecting congestion control for consistent high performance,'' in Proc. NSDI, Oakland, CA, USA, May 2015, pp. 395–408.

[10]. V. Arun and H. Balakrishnan, ''Copa: Practical delay-based congestion control for the Internet,'' in Proc. NSDI, Renton, WA, USA, 2018, pp. 329–342.

[11]. F.Ahmed,J.Erman,Z.Ge,A.X.Liu,J.Wang,andH.Yan,''Detectingand localizing end-to-end performance degradation for cellular data services,'' inProc.IEEEINFOCOM-35thAnnu.IEEEInt.Conf.Comput.Commun., Portland, OR, USA, Apr. 2016, pp. 459–460.

[12]. Modified DummyNet. Accessed: Sep. 1, 2020.[Online]. Available: https://github.com/mclab-cuhk/netmap-ipfw [46] Y.-C. Chen, E. M. Nahum, R. J. Gibbens, and D. Towsley, ''Measuring cellular networks: Characterizing 3G, 4G, and path diversity,'' in Proc. Annu. Conf. Int. Technol. Alliance, Jun. 2012, pp. 1–8.

[13]. Bonree. Accessed: Sep. 1, 2020.[Online]. Available: https://www.bonree. Com

[14]. G. C. Kessler. An Overview of Cryptography. Accessed: Sep. 1,2020. [Online]. Available: http://www.garykessler.net/library/crypto.html