Journal of Electronic Research and Application, 2024, Volume 8, Issue 2
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

Congestion Control Algorithms for the Internet – A Secondary Publication

Satoshi Utsumi*

Fukushima University, Fukushima, Japan

***Corresponding author:** Satoshi Utsumi, sutsumi@gmail.com

Abstract: In the last five years, there has been a V-shaped recovery in the number of papers on congestion control algorithms on the Internet. In this paper, congestion problems on the Internet are discussed, such as congestion collapse and bufferbloat from the perspective of the necessity of congestion control algorithms. The typical congestion control algorithms are introduced, and the research areas and methods of congestion control algorithms are described. Recent research trends and future prospects of congestion control algorithms are also presented.

Keywords: TCP; Congestion control algorithm; Congestion; Internet

Online publication: March 29, 2024

1. Introduction

The number of papers on congestion control algorithms on the Internet has shown a V-shaped recovery in the last five years. In IEEE Xplore, the keyword search “Internet congestion control” yielded 1,577, 1,247, 795, and 1,117 hits from 2002 to 2006, 2007 to 2011, 2012 to 2016, and 2017 to 2021, respectively [1]. The period from 2012 to 2016 was a winter period for research on congestion control algorithms. The last five years have seen a resurgence in research on congestion control algorithms worldwide, partly due to the emergence of BBR (bottleneck bandwidth and round-trip propagation time) announced by Google Inc [2–4].

This paper begins by defining congestion and its significance in prompting the development of congestion control algorithms. Various congestion-related challenges in the Internet, such as congestion collapse and buffer bloat are then described. Following this, a historical overview of congestion control algorithms within the context of Internet development is provided. This section highlights the evolution of approaches adopted to mitigate congestion. Furthermore, this paper outlines prevalent research areas and methodologies employed in the study and implementation of congestion control algorithms. Moreover, recent research trends and future prospects regarding congestion control algorithms are discussed, offering insights into emerging directions and potential challenges. The paper concludes by summarizing key insights gleaned from the discussion and drawing conclusions regarding the current state and future outlook of congestion control algorithms.

2. Congestion and congestion control algorithms

2.1. Congestion (e.g. traffic)

To understand why congestion control algorithms are needed in the first place, this section defines congestion and discusses historically important congestion problems in the Internet.

2.1.1. Definition of congestion

In this paper, congestion is defined as the occurrence of a packet queue or packet drop in the buffer of the bottleneck link when the input load to the bottleneck link on the communication path exceeds the link's capacity (bandwidth). When the input load to the bottleneck link exceeds the bandwidth, the size of the packet queue in the bottleneck link's buffer may exceed the buffer size, leading to buffer overflow and packet drop. Congestion leads to degradation in performance metrics such as throughput, latency, and packet drop rate.

2.1.2. Congestion collapse

Transmission control protocol (TCP) is a protocol designed to facilitate highly reliable communication between terminals [5]. The TCP described in RFC (Request for Comments) 793, issued in 1981, did not include congestion control algorithms but implemented flow control, as shown in **Figure 1**. [5]. Flow control involves regulating the amount of data transmitted by the sender based on the receiver's capacity to manage incoming data. Specifically, flow control in TCP is achieved by the receiver notifying the sender of the buffer size, indicating the amount of data it can receive, through acknowledgment (ACK) packets. The TCP receiver sends an ACK packet for each received data packet. In other words, in RFC 793, TCP's flow control ensures that the amount of data in flight (i.e., the data packets sent by the TCP sender but not yet acknowledged by ACK packets) matches the buffer size of the TCP receiver as indicated by the ACK packets. However, this method led to a phenomenon known as congestion collapse. Congestion collapse, as described in literature, occurs due to congestion-induced retransmissions (referred to as retransmissions here) that result from congestion [6,7].

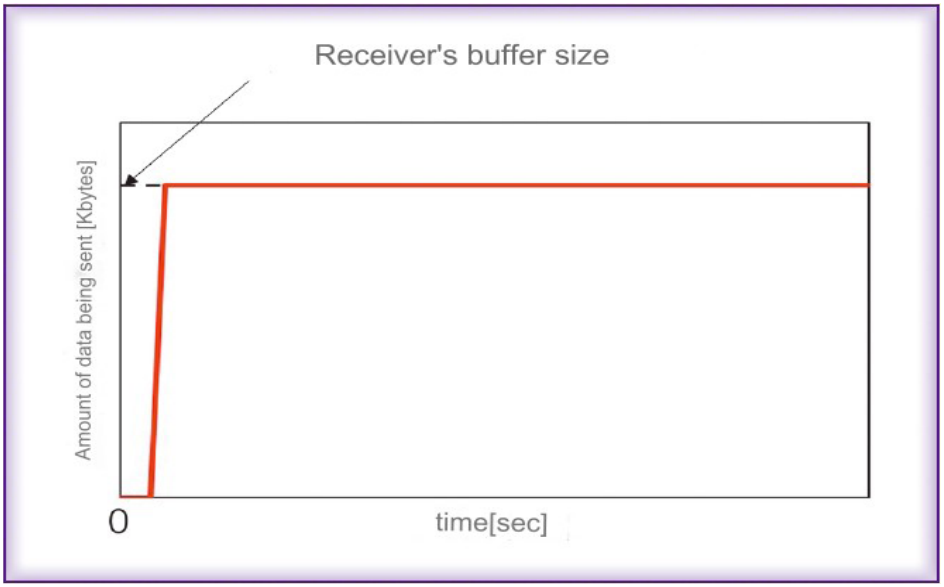

Figure 1. RFC 793 flow control

When severe congestion occurs, leading to packet drops or significant buffering delays, TCP resends data packets for which ACK packets have not arrived within a certain time frame. Buffering delay refers to delays caused by packet queues in bottleneck links. The time elapsed from sending a data packet until its

corresponding ACK packet is received and the subsequent initiation of retransmission is termed retransmission timeout (RTO). Since TCP in RFC 793 implements flow control without a congestion control algorithm, upon RTO, it resends all data packets that have timed out sequentially. Due to the imprecise calculation of RTO in RFC 793, data packets may be resent even if packet drops have not occurred. Consequently, congestion worsens as successive retransmitted packets accumulate in the bottleneck link, leading to a decrease in throughput. This phenomenon is known as congestion collapse. Reports indicated that during a congestion collapse in October 1986, the throughput between Lawrence Berkeley Laboratory (LBL) and the University of California, Berkeley (UC Berkeley) decreased from 32 kbit/s to 40 bit/s [8].

To address congestion collapse, the loss-based congestion control algorithm was devised, which reduces the packet transmission rate upon detecting packet loss to alleviate congestion. In Section 2.2, we introduce Tahoe, Reno, and CUBIC as examples of loss-based congestion control algorithms.

2.1.3. Bufferbloat

The typical loss-based congestion control algorithm increases the packet transmission rate until it detects packet loss. Upon detecting packet loss, it decreases the packet transmission rate to alleviate congestion.

In recent years, due to the decreasing cost of memory, there has been a trend towards increasing buffer sizes in nodes such as routers and switches connected to networks. Increasing buffer sizes has the effect of reducing packet discard rates and avoiding decreased bandwidth utilization rates.

On the other hand, as shown in **Figures 2 and 3**, the magnitude of buffering delays for flows controlled by loss-based congestion control algorithms like Reno or CUBIC depends on the size of the buffers at the bottleneck link. When the buffers at the bottleneck link, where flows controlled by loss-based congestion control algorithms exist, are large, the buffering delay at that bottleneck link increases. Depending on the bandwidth and buffer size at the bottleneck link, buffering delays lasting several hundred milliseconds or seconds may persist. This is particularly detrimental for applications that prioritize real-time performance, such as video conferencing systems like Zoom or Microsoft Teams, online multiplayer games, and remote control/surveillance applications. This phenomenon in the Internet is referred to as Bufferbloat and has been observed since around 2009 [9-11].

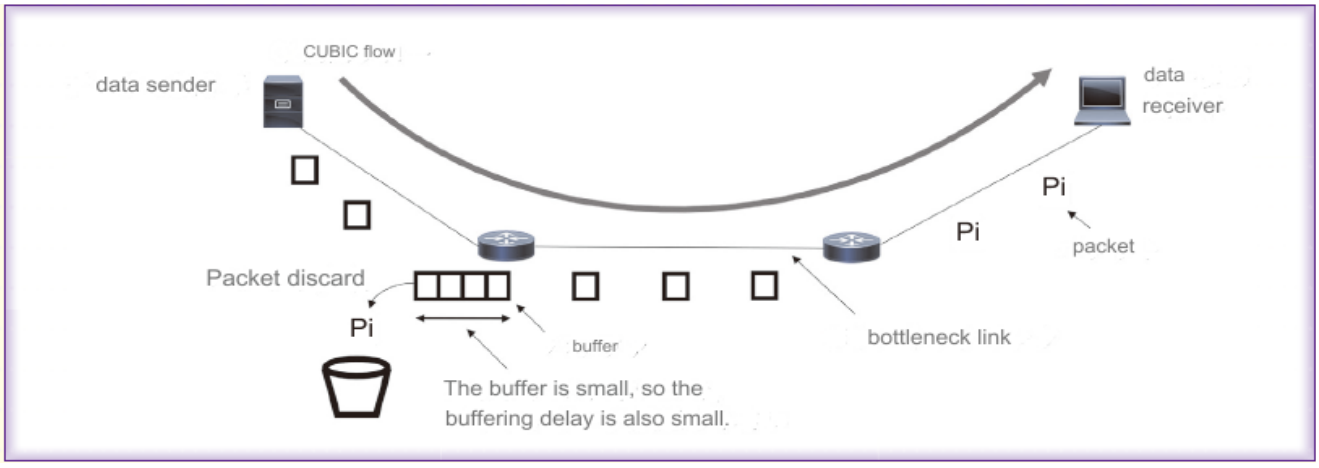

Figure 2. When the buffer at the bottleneck link is small

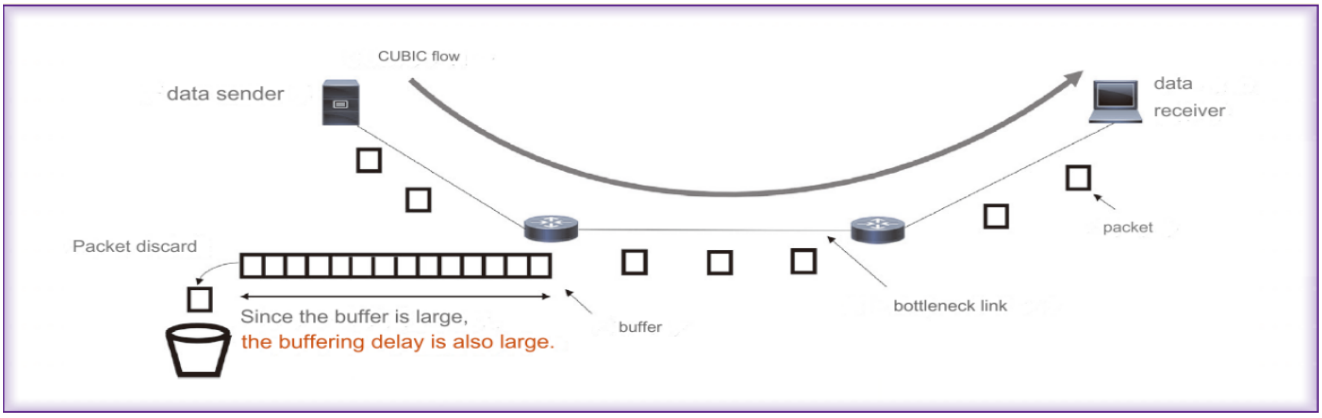

Figure 3. When the buffer at the bottleneck link is large

2.2. Congestion control algorithm

When we refer to congestion control algorithms, we generally mean the methods of adjusting the packet transmission rate between endpoints. In this section, we will focus on congestion control algorithms that have been widely used on the Internet, introducing algorithms aimed at resolving congestion collapse and bufferbloat.

2.2.1. Tahoe

Tahoe is an early version of the loss-based congestion control algorithm developed by Van Jacobson in 1988 [8]. Here, we provide explanations of basic terminologies related to congestion control algorithms. Round-trip time (RTT) refers to the time it takes for a sender to transmit a data packet and receive the corresponding ACK packet. Congestion window refers to the parameter that adjusts the amount of data in transit among the packets sent but not yet acknowledged by ACK packets. In other words, the congestion window regulates the amount of data in transit. Typically, congestion control algorithms adjust the packet transmission rate by controlling the congestion window.

Figure 4 illustrates the change in the congestion window of Tahoe. At the start of the connection (time 0 (s)), the congestion window begins with a data volume equivalent to one packet and undergoes exponential growth during a slow start. Here, it's assumed that a finite value initializes the threshold. When the congestion window reaches the threshold, it transitions to congestion avoidance mode, where it increases linearly. Upon detecting packet loss, the congestion window resets to the size of one packet, and a slow start is executed again. At this point, the threshold is set to half the size of the congestion window when packet loss is detected. Tahoe was once included in Linux 1.0 but is not currently part of the Linux kernel.

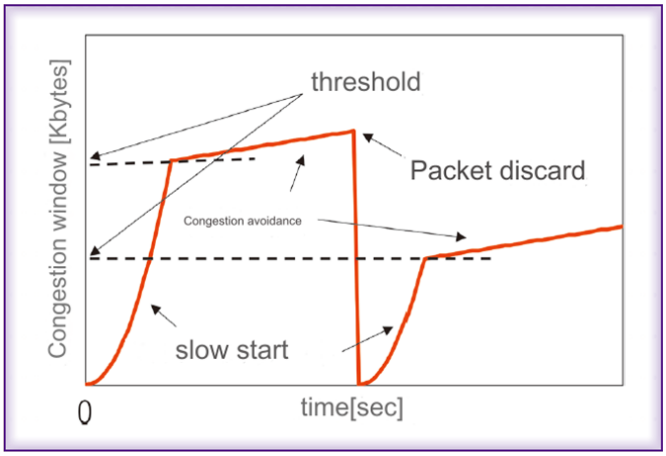

Figure 4. Changes in Tahoe's congestion window

Tahoe made it possible to avoid congestion collapse. However, Tahoe introduced a new challenge of decreased utilization of bottleneck bandwidth because it resets the congestion window to the size of one packet and executes a slow start each time packet loss is detected.

2.2.2. Reno

Sections 2.2.2 through **2.2.5** describe the behavior of congestion control algorithms using the network emulator Mininet [12]. Specifically, the network topology depicted in Figure 5 is configured on Mininet, with two transmission hosts sending TCP flows controlled by the same congestion control algorithm, one flow each. In other words, transmission host S1 sends a TCP flow to receiving host R1, while S2 sends a TCP flow to R2 as illustrated in **Figure 5**. The bottleneck bandwidth is set to 50 Mbit/s, the round-trip propagation delay between hosts is 40 ms, and the buffer size at the bottleneck link is 1 MByte. The observation period is set to 120 s. Each section presents the time evolution of the amount of data transmitted and the RTT of the TCP flow sent from S1 according to the congestion control algorithm explained in that section.

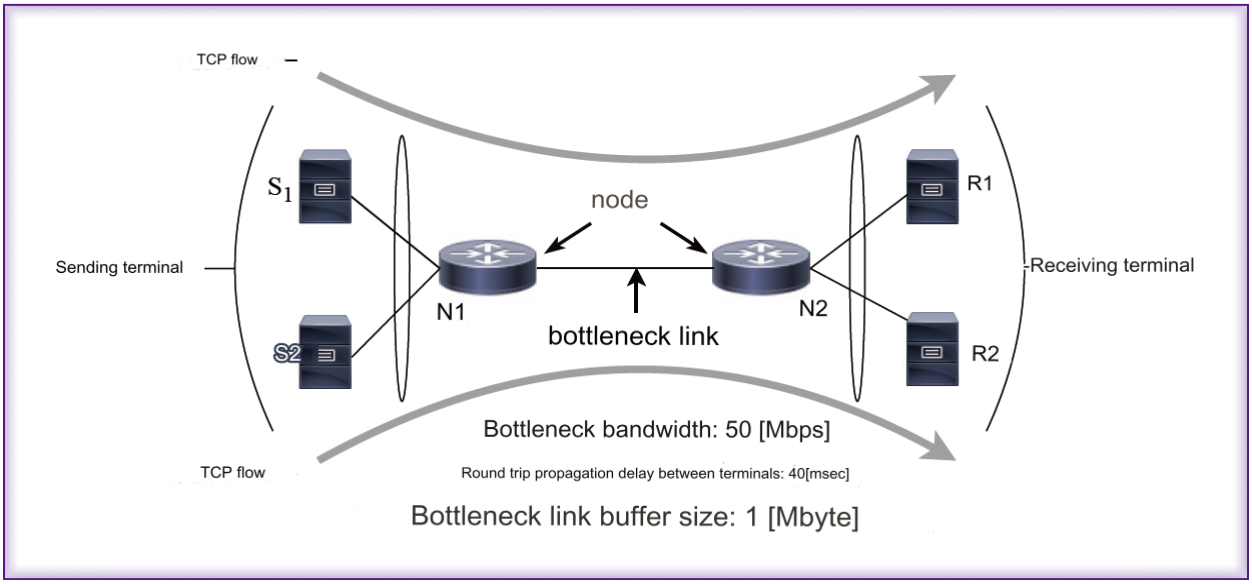

Figure 5. Network topology

Reno is an improved version of Tahoe, which is a loss-based congestion control algorithm [13]. **Figures 6–7** respectively illustrate the time evolution of the amount of data transmitted and the RTT for Reno flows. From **Figure 6**, we observe that immediately after the peak of the transmitted data amount, indicating the detection of packet loss, the amount of data being transmitted reduces by half. Subsequently, it linearly increases again. This behavior in Reno occurs because, fundamentally, it does not revert to a slow start after detecting packet loss but rather halves the congestion window and repeats congestion avoidance. Reno adjusts the congestion window using AIMD (additive-increase/multiplicative-decrease) based on information from received ACK packets and detected packet loss. Compared to Tahoe, Reno reduces the congestion window by a smaller margin upon detecting packet loss, thereby improving the utilization of bottleneck bandwidth. From Figure 7, we can observe that the RTT for Reno flows linearly increases similarly to the amount of data transmitted and decreases after the occurrence of packet loss. The extent of the decrease depends on the proportion of Reno flows that detect packet loss. Reno has been implemented in Linux 1.3.90 [14], as well as in Windows 95/NT.

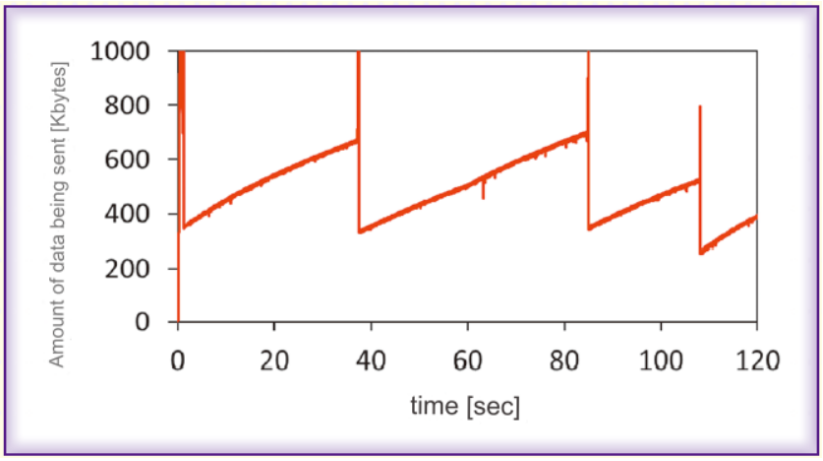

Figure 6. Changes in the amount of data being sent by Reno

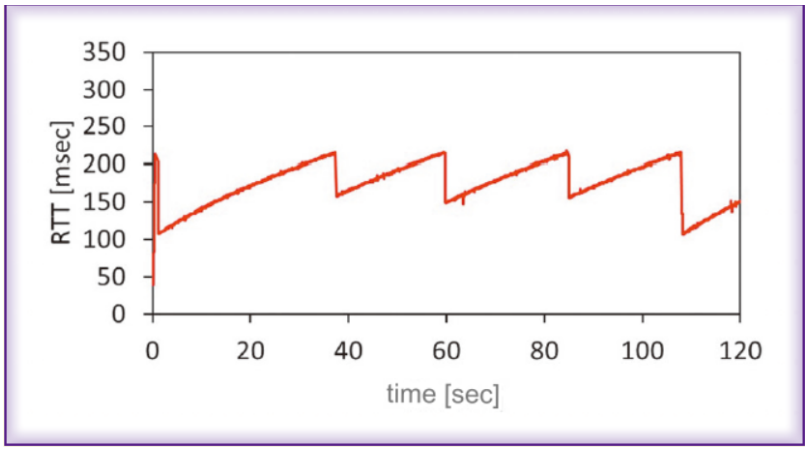

Figure 7. Reno RTT change

2.2.3. CUBIC

CUBIC is another loss-based congestion control algorithm. **Figures 8 and 9** show the time variation of the amount of data being transmitted and the RTT of the CUBIC flow, respectively [15]. **Figure 8** shows that the amount of data being transmitted increases cubically then decreases significantly immediately after its peak, and then increases cubically again. Generally, CUBIC reduces the congestion window to 70% of its size upon detecting packet loss. CUBIC is known to achieve high utilization rates even in networks with large bottleneck bandwidth. Additionally, the "Fast Convergence" feature allows CUBIC to implement two different cubic increase methods for congestion avoidance. In **Figure 8**, two cubic shapes with different heights based on turning points determined by different calculation methods are observed. Fast Convergence aims to improve the convergence speed of throughput between CUBIC flows. Since CUBIC only reduces the congestion window to 70% upon detecting packet loss, compared to Reno, which halves it, there tend to be larger buffering delays in paths with large buffer sizes at bottleneck links. Compared to the RTT of Reno shown in **Figure 7**, the RTT of CUBIC in **Figure 9** is generally larger. CUBIC has been implemented in Linux 2.6.16 and Windows 10.

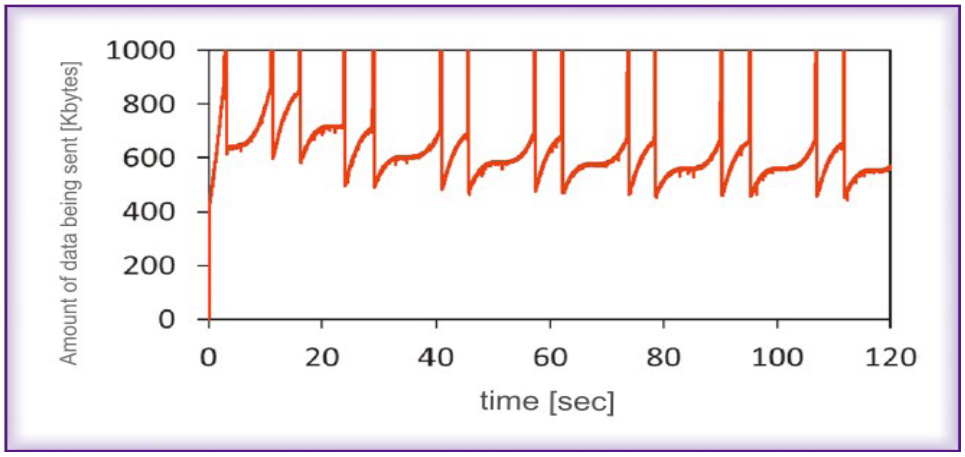

Figure 8. Changes in the amount of data being sent by CUBIC

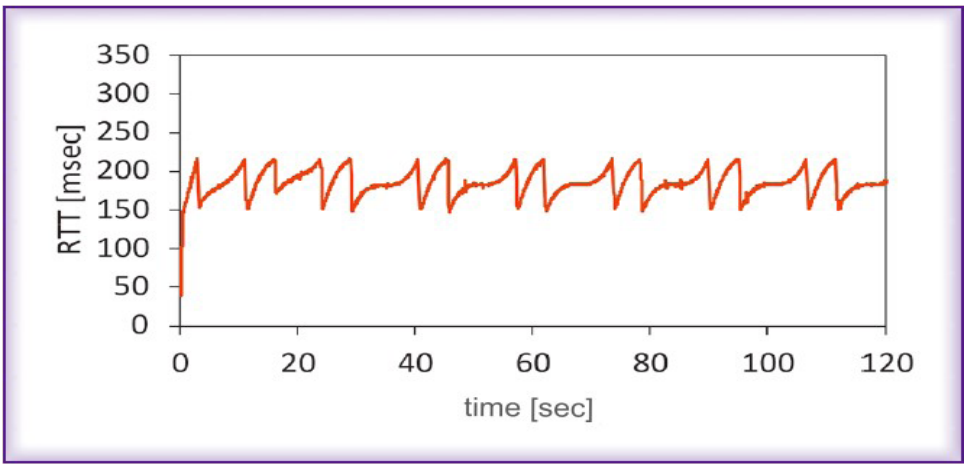

Figure 9. Change in RTT of CUBIC

2.2.4. BBR

As shown in **Section 2.1.3**, loss-based congestion control algorithms such as Reno and CUBIC can cause bufferbloat when the buffer at the bottleneck link is large. 2016, Google introduced BBR [4]. BBR is a congestion control algorithm aimed at reducing buffering delay at bottleneck links, referred to as congestion-based congestion control. BBR aims to reach the optimal operating point, achieving maximum throughput and minimum RTT, by iterating through two phases: ProbeBW and ProbeRTT. In the ProbeBW phase, it temporarily increases the packet transmission rate to explore maximum throughput, while in the ProbeRTT phase, it significantly reduces the congestion window to explore minimum RTT. Using the values of the maximum throughput and minimum RTT obtained during these explorations, BBR determines the packet transmission rate and congestion window size. BBR has been observed to operate near the optimal operating point when occupying the bottleneck link with a single flow. However, when multiple flows share the bottleneck link, it operates away from the optimal operating point, leading to increased buffering delay and RTT [16].

Figures 10–11 show the time variation of the amount of data being transmitted and the RTT of a BBR flow, respectively; the decrease in the amount of data being transmitted and the RTT every 10 seconds or so indicates the time when the ProbeRTT phase is executed. By comparing **Figure 11** with **Figures 7 and 9**, it is evident that the RTT of BBR is smaller than that of Reno and CUBIC. This suggests a certain degree of success in mitigating buffering delay. BBR has been integrated into Linux since version 4.9 and is used on Google's own servers, such as YouTube servers, as well as on AWS (Amazon Web Services).

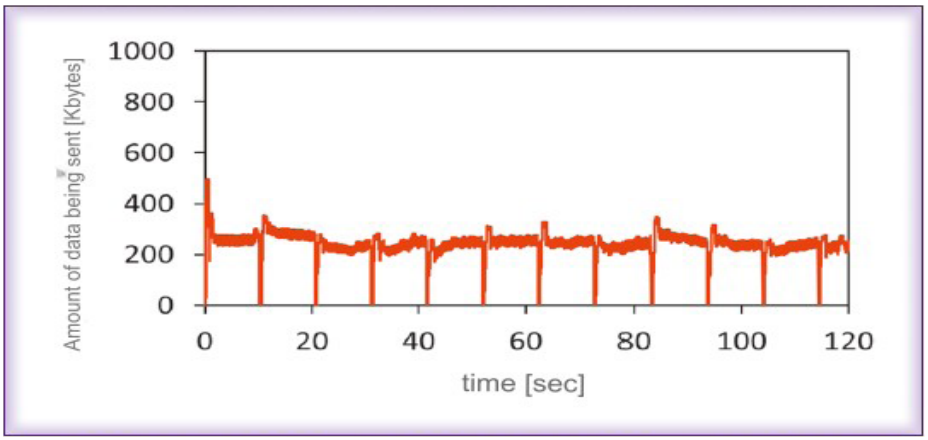

Figure 10. Changes in the amount of data being sent by BBR

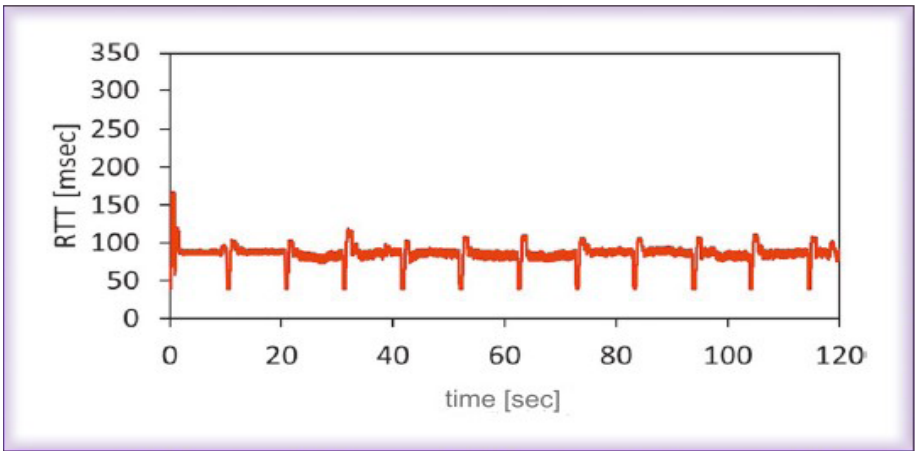

Figure 11. BBR RTT changes

2.2.5. Copa

Copa is a delay-based congestion control algorithm based on queueing theory [17,18]. Copa aims to maximize the objective function $U = \log(\lambda) - \delta * \log(d)$, where λ (packet/s), d (s), and δ ($0 < \delta \leq 1.0$) represent throughput, buffering delay, and Copa's parameter, respectively. Specifically, Copa attempts to achieve this objective by determining the packet transmission rate λ as $1 / (\delta * d)$ (packet/s) in response to the observed buffering delay d. The standard value for δ is typically $\delta = 0.5$.

Copa has a competitive mode designed for sharing flows with loss-based congestion control algorithms and bottleneck links. When a Copa flow cannot observe the recent minimum RTT, it switches to the competitive mode, assuming it shares the bottleneck link with a loss-based congestion control flow. In the competitive mode, Copa adjusts the value of δ using additive-increase/multiplicative-decrease (AIMD) based on ACK packet reception and packet loss detection, aiming to minimize $1/\delta = 2$.

Copa aims to improve fairness in throughput with flows using loss-based congestion control algorithms like Reno or CUBIC by actively sending packets more aggressively than a Copa flow where $1/\delta$ is fixed and adjusted using AIMD with $\delta = 0.5$.

Figures 12 and 13 show the time variation of the amount of data being sent and the RTT of the Copa flow, respectively. The total throughput and average RTT of the two flows, Reno, CUBIC, BBR, and Copa, are shown in **Figure 14**, based on the data observed in **Sections 2.2.2** through **2.2.5**. **Figure 14** shows the total throughput

and average RTT for the Reno, CUBIC, BBR, and Copa flows, respectively. **Figure 14** shows that the total throughput of the Copa flow is almost the same as that of the flows using other congestion control algorithms, while the average RTT is significantly improved.

Copa has been implemented in CCP (congestion control plane) [19] and mvfst [20], and Meta (formerly Facebook) uses Copa for uploading live video on Android devices.

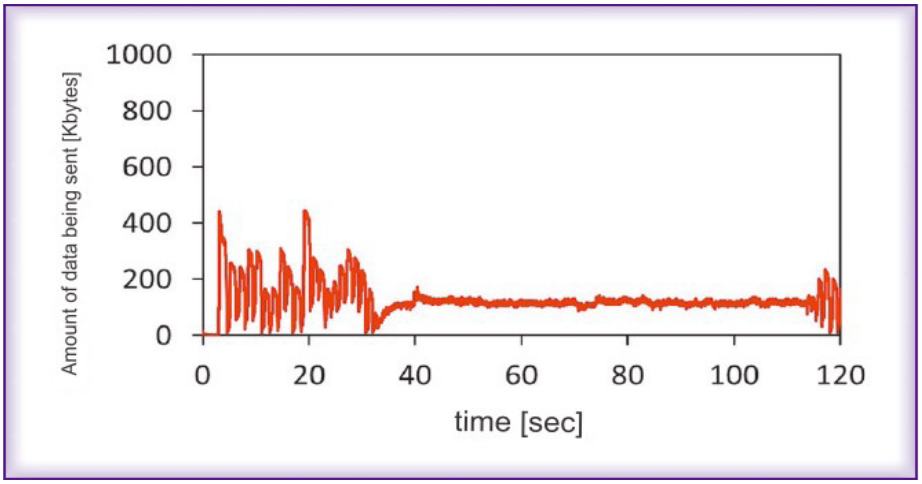

Figure 12. Changes in the amount of data being sent by Copa

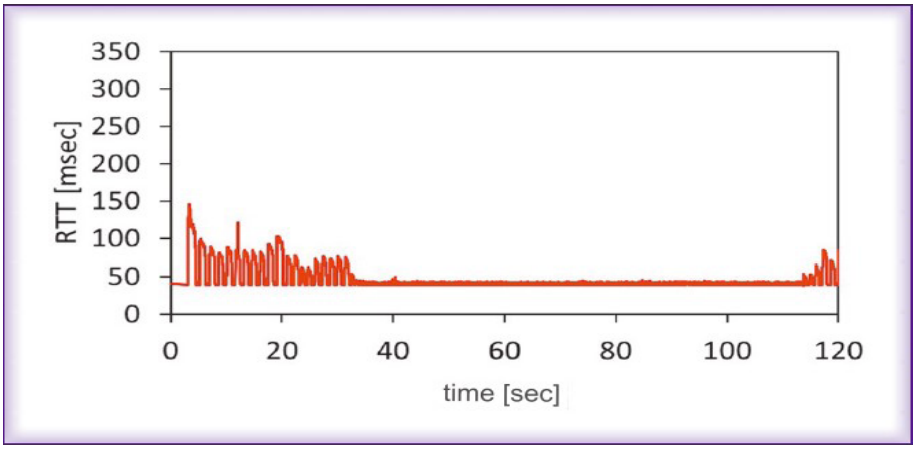

Figure 13. Change in RTT of Copa

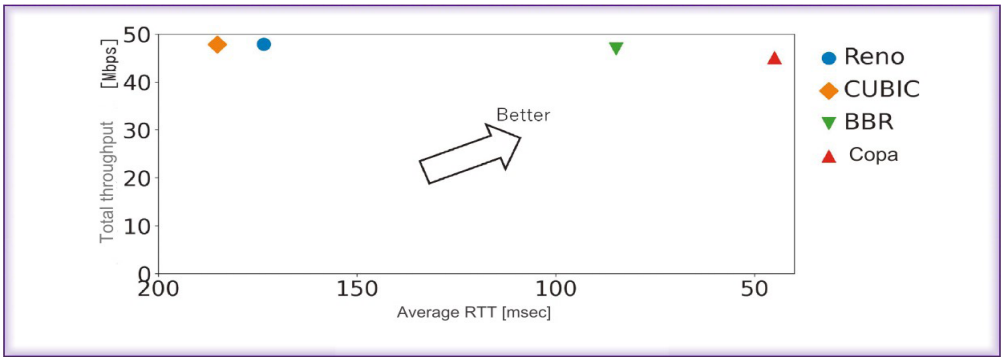

Figure 14. Total throughput and average RTT

3. Research area of congestion control algorithms

Congestion control algorithms have been incorporated into transport layer protocols such as TCP, QUIC [21], and datagram congestion control protocol (DCCP) [22]. Much of the research on congestion control algorithms focuses on how to implement high-performance, fair, or efficient methods for adjusting packet transmission rates. In congestion control algorithm research, the following metrics are commonly used and evaluated.

(1) Throughput

Throughput indicates how much data is transmitted in a unit of time. Instantaneous throughput is considered in some cases, while in others, only average throughput is considered. Throughput is calculated by excluding duplicate data received. A higher throughput value indicates better performance. Throughput is typically measured in units such as gigabits per second (Gbit/s), megabits per second (Mbit/s), or packets per second (packet/s).

(2) Throughput

Latency refers to the time taken for packet transmission and reception between terminals, including one-way delay and round-trip time (RTT). Generally, terminal-to-terminal latency consists of propagation delay, processing delay, and buffering delay. Propagation delay is the time taken for the physical signal of a packet to pass through the transmission medium and reach the next node. Processing delay refers to the time taken for packet forwarding processing at a node, including routing and switching processes. Among these, minimizing buffering delay is one criterion for evaluating the effectiveness of congestion control algorithms. Reducing delay is desirable, and it is typically measured in units such as nanoseconds (ns), microseconds (μs), milliseconds (ms), or seconds (s).

(3) Transfer time

Transfer time is the time from the start of transmission to the completion of transmission of data of a certain size. It is equal to the data size divided by the average throughput from the start of transmission to the completion of transmission. A smaller transfer time indicates better performance. Transfer time is typically measured in units such as seconds, minutes, or hours.

(4) Fairness

Fairness is a metric, devised by Raj Jain, that indicates how equitably flows sharing a bottleneck link are dividing bandwidth [23]. It is also known as Jain's fairness index. Fairness is represented by a value between 0 and 1, where a value closer to 1 indicates better fairness. It is a unitless measure.

(5) Packet drop rate

Indicates how many packets are dropped in packet transmission and reception between terminals. The packet drop rate takes a value between 0 and 1, with a value closer to 0 indicating a good condition. The unit is none or %.

In recent years, an indicator called "harm" [24] has been proposed to demonstrate how much impact a flow sharing the bottleneck link has on other flows. "Harm" is expected to range from 0 to 1, with smaller values indicating better conditions. The specific calculation method for "harm" is detailed in several papers [24,25].

Additionally, an indicator called "power" [26], which divides throughput by latency, has been proposed by the queueing theory expert Leonard Kleinrock. The value of "power" indicates better conditions when it is larger.

Node support for terminal-to-terminal congestion control algorithms includes active queue management (AQM) [27,28] and explicit congestion notification (ECN) at the network layer [29], and research on these methods has been conducted extensively.

4. Research methodology for congestion control algorithms

As described in Section 3, research on congestion control algorithms is often aimed at improving the communication performance between terminals. More specifically, research is commonly conducted in the following manner:

4.1. Proposed congestion control algorithm

The objective of this study is to propose a new congestion control algorithm or improvement method that is more efficient, fair, or effective than the previous ones in a given network environment.

4.2. Simulation experiments

The performance of the previously proposed and newly proposed congestion control algorithms can be evaluated using network simulators. Free simulators such as ns-2 [30] and ns-3 [31] are commonly used.

4.3. Emulation experiments

Emulation typically involves evaluating the performance of previously proposed congestion control algorithms as well as newly proposed ones using network emulators. Traditionally, experiments were often conducted by connecting multiple computers to configure a network and using network emulators such as dummynet [32] and traffic control (tc) [33]. However, recently, it has become more common to conduct evaluations using tools like Mininet [12] and Mahimahi [34] on a single computer. With Mininet, it is possible to create virtual network topologies using tc. Additionally, Mininet supports the use of active measurement tools like iPerf [35].

4.4. Experiments in real-world environments

Experiments in real-world environments involve evaluating the performance of previously proposed congestion control algorithms as well as newly proposed ones in real network environments. This can be done by conducting experiments in proprietary network environments such as company networks or by utilizing publicly available experimental servers (such as iPerf servers [36]) or cloud platforms (like Amazon EC2 [37]) via the internet.

4.5. Mathematical performance analysis

The experimentation involves determining the performance metrics such as throughput and transfer time analytically for the proposed congestion control algorithms as well as existing ones. This not only serves the purpose of performance evaluation but also aids in proposing new congestion control algorithms [38,39], such as those ensuring fair performance along with the flow controlled by the congestion control algorithms analyzed.

5. Recent research trends and prospects

This section describes recent research trends and prospects for congestion control algorithms.

5.1. Recent research trends

5.1.1. Survey of congestion control algorithms share on the Internet

In Mishra et al., study [40], a tool was developed to estimate the congestion window corresponding to consecutive RTTs at the receiver side of TCP. Additionally, an offline analyzer was constructed to identify TCP congestion control algorithms based on the time-series tracking history of congestion windows. Using these tools, an investigation into the usage of TCP congestion control algorithms revealed that approximately 36% of websites

employ CUBIC, while around 22% use BBR. It was estimated that BBR accounts for over 40% of the total traffic on the Internet.

5.1.2. Application of machine learning to congestion control algorithms [41-48]

Remy [41] optimizes and generates congestion control algorithms offline based on pre-specified congestion control objectives, protocol assumptions, and models of both network and traffic. Performance-oriented congestion control (PCC) [42] sends packets at two different rates, slightly higher and slightly lower than the current rate. If either of these rates performs better, it is selected as the next packet transmission rate, and this direction continues as long as performance improves. This online learning approach helps overcome the limitations of Remy's offline optimization, where performance may degrade if input assumptions and network models deviate from the actual network environment. Recently, practical hybrid approaches combining classic congestion control methods with advanced deep reinforcement learning techniques have emerged [48].

5.1.3. Performance evaluation and analysis model for multiple flows in core link

The conventional congestion control algorithm analysis models implicitly assumed congestion occurred only at the edge links. In other words, evaluations of congestion control algorithms were limited to scenarios with tens of flows and bandwidth scales of several hundred Mbit/s. However, in recent years, it has become known that congestion can also occur in core links with thousands of flows and bandwidths ranging from 1 to several hundred Gbit/s. In Philip et al.'s work [49], it was revealed that the analysis model of NewReno, derived from assuming congestion at edge links, does not hold in core links. Furthermore, it was shown that BBR flows, which demonstrate good fairness at edge links, may become highly unfair in core links.

5.1.4. Reverse engineering of congestion control algorithms

The internal structure of congestion control algorithms in non-open source operating systems may be unknown. Ferreira *et al.* [50] proposed a method for reverse engineering congestion control algorithms. This method derives a synthesis program based on the observed behavior of the original congestion control algorithm. By implementing the synthesized program, the congestion control algorithm can be evaluated.

5.1.5. Competing heterogeneous congestion control algorithms [39, 51,52]

Studies on competition between different congestion control algorithms at bottleneck links have been conducted since around 2000 [53,54]. However, with the emergence of new congestion control algorithms like BBR and Copa, such research has continued to be active in recent years. In Ware *et al.*'s work [51], the interaction between flows using loss-based congestion control algorithms such as CUBIC and Reno and BBR flows is analyzed and verified through experiments. In another research [39], an improved version of BBR that improves throughput fairness when BBR and CUBIC flows contend for the bottleneck link was proposed and evaluated using an analytical method. Goyal *et al.* [52] proposed and evaluated the congestion control algorithm NimbusCC based on a method for detecting whether cross-traffic is elastic. Here, "elastic" behavior refers to increasing packet transmission rates when more bandwidth is detected to be available and decreasing them otherwise. When sharing bottleneck links with flows using elastic congestion control algorithms, NimbusCC operates in TCP-competitive mode, behaving similarly to CUBIC or NewReno. When sharing bottleneck links only with non-elastic cross-traffic, NimbusCC operates in delay-controlling mode, achieving low buffering delay using algorithms such as Vegas, Copa (non-competitive mode), or BasicDelay inspired by the explicit control protocol (XCP) [55-57].

5.2. Prospects

In the past, performance evaluation of congestion control algorithms was predominantly conducted in uniform environments using simulators or emulators. In recent years, there has been some effort to evaluate the performance of congestion control algorithms in Internet environments, but the evaluation environments are not yet sufficiently established. Although experimental iPerf servers [36] are available, the number of servers that are actually operational and usable is limited. Furthermore, while Pantheon [58] was introduced in 2018 with the expectation of providing a comprehensive evaluation environment for congestion control algorithms on the Internet, support for it has already been discontinued. Recently, issues such as extreme unfairness between flows using the same congestion control algorithm in real environments, as highlighted by Starvation [59], have underscored the increasing importance of performance evaluation and behavioral analysis of congestion control algorithms in practical environments. There is a growing need for the development of congestion control algorithms that truly perform well and fairly on the Internet based on performance evaluations in environments where congestion control algorithms can be evaluated on a large scale [60].

6. Conclusion

In this paper, congestion issues such as congestion collapse and Bufferbloat were discussed from the perspective of why congestion control algorithms are necessary on the Internet. Additionally, representative congestion control algorithms in the history of the Internet were introduced, and the research areas and methodologies of congestion control algorithm studies were discussed. Furthermore, recent research trends and future prospects of congestion control algorithms were addressed. This paper serves to contribute to the emergence and growth of young researchers in the field of congestion control algorithms on the Internet.

Funding

This work was supported by JSPS Grants-in-Aid for Scientific Research JP20K11786 and JP21KK0202. We thank Mai Komatsubara and Keisuke Kano of Fukushima University for reviewing the final manuscript.

Disclosure statement

The author declares no conflict of interest

References

[1] IEEE Xplore, 2022, https://ieeexplore.ieee.org/Xplore/home.jsp

[2] Hasegawa T, 2020, Internet Congestion Control: Past and Future. Communications Technology Journal, 2020: 33–38.

[3] Web of Science, 2022, https://www.webofknowledge.com

[4] Cardwell N, Cheng Y, Gunn CS, et al., 2017, BBR: Congestion-Based Congestion Control, Commun. ACM, 60(2): 58–66.

[5] Postel J, (ed) 1981, Transmission Control Protocol (RFC 793), IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc793

[6] Nagle J, 1984, Congestion Control in IP/TCP Internetworks (RFC 896), IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc896#:~:text=of%20that%20standard.-,RFC%20896%20Congestion%20Control%20in%20IP%2FTCP%20Internetworks%201%2F6,window%20size%20has%20been%20reduced

[7] Afanasyev A, Tilley N, Reiher P, 2010, Host-to-Host Congestion Control for TCP. IEEE Communications Surveys & Tutorials, 12(3): 304–342, .

[8] Jacobson V, Karels MJ, 1988, Symposium Proceedings on Communications Architectures and Protocols, August 16–18, 1988: Congestion Avoidance and Control. Stanford, 314–329.

[9] Zoom, n.d., https://zoom.us/signin

[10] Microsoft Teams, n.d., https://www.microsoft.com/ja-jp/microsoft- teams/log-in

[11] Gettys J, 2011, Bufferbloat: Dark Buffers in the Internet. IEEE Internet Computing, 15(3): 96.

[12] Mininet: An Instant Virtual Network on Your Laptop (or Other PC), 2022, http://mininet.org/

[13] Allman M, Paxson V, Stevens W, 1999, TCP Congestion Control (RFC5681), IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc5681

[14] Henderson T, Floyd S, Gurtov A, et al., 2012, The NewReno Modification to TCP's Fast Recovery Algorithm (RFC 6582), IETF Datatracker, https://datatracker.ietf.org/doc/rfc6582/

[15] Ha S, Rhee I, Xu L, 2008, CUBIC: A New TCP-Friendly High-Speed TCP Variant. ACM SIGOPS Operating Systems Review, 42(5): 64–74.

[16] Hock M, Bless R, Zitterbart M, 2017, 2017 IEEE 25th International Conference on Network Protocols (ICNP), October 10–13, 2017: Experimental Evaluation of BBR Congestion Control, 1–10

[17] Arun V, Balakrishnan H, 2018, Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18), April 9–11, 2018: Copa: Practical Delay-Based Congestion Control for the Internet. Renton, 329–342.

[18] Kleinrock L, 1975, Queueing Systems Volume 1: Theory, Wiley-Interscience, New York.

[19] Narayan A, Cangialosi F, Raghavan D, et al., 2018, Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, August 20–25, 2018: Restructuring Endpoint Congestion Control. Budapest, 30–43.

[20] Facebook's QUIC Implementation, https://github.com/facebookincubator/mvfst, 2022.

[21] Iyengar J, Thomson M, 2021, QUIC: A UDP-Based Multiplexed and Secure Transport (RFC 9000), Datatracker, https://datatracker.ietf.org/doc/rfc9000/

[22] Kohler E, Handley M, Floyd S, 2006, Datagram Congestion Control Protocol (DCCP) (RFC4340), Datatracker, https://datatracker.ietf.org/doc/html/rfc4340

[23] Jain R, Durresi A, Babic G, 1999, Throughput Fairness Index: An Explanation, https://www.cse.wustl.edu/~jain/atmf/ftp/atm99-0045.pdf

[24] Ware R, Mukerjee MK, Seshan S, et al., 2019, HotNets '19: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, November 13–15, 2019: Beyond Jain's Fairness Index: Setting the Bar for the Deployment of Congestion Control Algorithms. New York, 17–24.

[25] Utsumi T, 2022, Congestion Control Algorithm and Fairness (or Affinity) with Analytical Model, Technical Report, NS2022-23, 1–5.

[26] Kleinrock L, 2018, Internet Congestion Control Using the Power Metric: *Keep the Pipe Just Full, But No Fuller*. Ad Hoc Networks, 80: 142–157.

[27] Adams R, 2012, Active Queue Management: A Survey. IEEE Communications Surveys & Tutorials, 15(3): 1425–1476.

[28] Nichols K, Jacobson V, McGregor A, et al., 2018, Controlled Delay Active Queue Management (RFC 8289), Datatracker, https://datatracker.ietf.org/doc/html/rfc8289

[29] Floyd S, 1994, TCP and Explicit Congestion Notification. ACM SIGCOMM Computer Communication Review, 24(5): 8–23.

[30] The Network Simulator-ns-2, n.d., https://www.isi.edu/nsnam/ns/

[31] ns-3 Network Simulator, n.d., https://www.nsnam.org/

[32] Rizzo L, 1997, Dummynet: A Simple Approach to the Evaluation of Network Protocols. ACM SIGCOMM Computer Communication Review, 27(1): 31–41.

[33] tc(8) - Linux Man Page, n.d., https://linux.die.net/man/8/tc

[34] Netravali R, Sivaraman A, Winstein K, et al., 2014, Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement, ACM SIGCOMM Computer Communication Review, 44(4): 129–130.

[35] iPerf - The Ultimate Speed Test Tool for TCP, UDP and SCTP, n.d., https://iperf.fr/

[36] Public iPerf3 Servers, n.d., https://iperf.fr/iperf-servers.php

[37] Amazon EC2, n.d., https://aws.amazon.com/jp/ec2/

[38] Floyd S, Handley M, Padhye J, et al., 2000, Equation-Based Congestion Control for Unicast Applications. ACM SIGCOMM Computer Communication Review, 30(4): 43–56.

[39] Utsumi S, Hasegawa G, 2022, 2022 IFIP Networking Conference (IFIP Networking), June 13–16, 2022: Improving Inter-Protocol Fairness Based on Estimated Behavior of Competing Flows. Catania, 1–9.

[40] Mishra A, Sun X, Jain A, et al., 2020, The Great Internet TCP Congestion Control Census. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 3(3): 45.

[41] Winstein K, Balakrishnan H, 2013, TCP ex Machina: Computer-Generated Congestion Control. ACM SIGCOMM Computer Communication Review, 43(4): 123–134.

[42] Dong M, Li Q, Zarchy D, et al., 2015, Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15), May 4–6: PCC: Re-Architecting Congestion Control for Consistent High Performance. Oakland, 395–408.

[43] Dong M, Meng T, Zarchy D, et al., 2018, Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18), April 9–11: PCC Vivace: Online-Learning Congestion Control. Renton, 343–356.

[44] Jay N, Rotman N, Godfrey B, et al., 2019, A Deep Reinforcement Learning Perspective on Internet Congestion Control. Proceedings of the 36th International Conference on Machine Learning, 3050–3059.

[45] Fang J, Ellis M, Li B, et al., 2019, Reinforcement Learning for Bandwidth Estimation and Congestion Control in Real-Time Communications. arXiv. https://doi.org/10.48550/arXiv.1912.02222

[46] Sivakumar V, Delalleau O, Rocktäschel T, et al., 2019, MVFST-RL: An Asynchronous RL Framework for Congestion Control with Delayed Actions. arXiv. https://doi.org/10.48550/arXiv.1910.04054

[47] Emara S, Li B, Chen Y, 2020, IEEE INFOCOM 2020 – IEEE Conference on Computer Communications, July 2020: Eagle: Refining Congestion Control by Learning from the Experts, 676–685.

[48] Abbasloo S, Yen CY, Chao HJ, 2020, Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, August 10–14, 2020: Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet, 632–647.

[49] Philip AA, Ware R, Athapathu R, et al., 2021, Proceedings of the 21st ACM Internet Measurement Conference, November 2–4, 2021: Revisiting TCP Congestion Control Throughput Models & Fairness Properties at Scale. Virtual, 96–103.

[50] Ferreira M, Narayan A, Lynce I, et al., 2021, Proceedings of the 20th ACM Workshop on Hot Topics in Networks, November 10–12, 2021: Counterfeiting Congestion Control Algorithms. United Kingdom, 132–139.

[51] Ware R, Mukerjee MK, Seshan S, et al., 2019, Proceedings of the Internet Measurement Conference, October 21–23, Amsterdam: Modeling BBR's Interactions with Loss-Based Congestion Control, 137–143.

[52] Goyal P, Narayan A, Cangialosi F, et al., 2022, Proceedings of the ACM SIGCOMM 2022 Conference, August 22–

26: Elasticity Detection: A Building Block for Internet Congestion Control. Amsterdam, 158–176.

[53] Hasegawa G, Murata M, Miyahara H, 1999, 18th IEEE Annual Joint Conference, March 21–25, 1999: INFOCOM, IEEE Computer and Communications Societies: Fairness and Stability of congestion Control Mechanisms of TCP. New York, 1329–1336.

[54] Hasegawa G, Kurata K, Murata M, 2000, Proceedings 2000 International Conference on Network Protocols, November 14–17, 2000: Analysis and Improvement of Fairness Between TCP Reno and Vegas for Deployment of TCP Vegas to the Internet. Osaka, 177–186.

[55] Goyal P, Agarwal A, Netravali R, et al., 2020, Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20), February 25–27: ABC: A Simple Explicit Congestion Controller for Wireless Networks. Santa Clara, 353–372.

[56] Katabi D, Handley M, Rohrs C, 2002, Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, August 19–23: Congestion Control for High Bandwidth-Delay Product Networks. Pittsburgh, 89–102.

[57] Tai C-H, Zhu J, Dukkipati N, 2008, The 27th Conference on Computer Communications, April 13–18, 2008: Making Largescale Deployment of RCP Practical for Real Networks. Pheonix, 2180–2188.

[58] Yan FY, Ma J, Hill GD, et al., 2018, 2018 USENIX Annual Technical Conference (USENIX ATC'18), July 11–13, 2018: Pantheon: The Training Ground for Internet Congestion Control Research, 731–743.

[59] Arun V, Alizadeh M, Balakrishnan H, 2022, Proceedings of the ACM SIGCOMM 2022 Conference, August 22–26, 2022: Starvation in End-To-End Congestion Control. 177–192

[60] Public-iPerf3-Serverlist, https://github.com/R0GGER/public-iperf3-servers