TCP-based Congestion Control Algorithms

Subjects: Telecommunications | Engineering, Electrical & Electronic | Computer Science, Information Systems
Contributor: Josip Lorincz, Zvonimir Klarin

In today's data networks, the main protocol used to ensure reliable communications is the transmission control protocol (TCP). The TCP performance is largely determined by the used congestion control (CC) algorithm, which main purpose is to avoid the congestion of the network that can be caused by a large burst of data traffic. TCP CC algorithms have evolved over the past three decades and a large number of CC algorithm variations have been developed to accommodate various communication network environments. Considering the importance of CC in communication networks, the fundamentals of the TCP as the main transport layer protocol and CC process have been explained in detail. Also, an encyclopedic overview of the most popular single-flow and multi-flow TCP CC algorithms with corresponding alternatives has been present. Future directions in the possible improvement of CC algorithms for implementation in complex heterogeneous networks composed of wired and wireless elements are lastly discussed in this encyclopedic work.

Keywords: TCP ; congestion control ; wireless ; wired ; network ; mobile ; algorithms ; communications ; Internet ; protocol

1. Introduction

At the transport layer of the Open System Interconnection (OSI) model, the transmission control protocol (TCP) is a major protocol used in today's communication over the Internet. The TCP is designed to provide a reliable end-to-end connection over unreliable links. Besides affecting the physical and media access control (MAC) layers of the OSI model, ensuring the high data rates in environments characterised with fluctuations in the quality of the wired links or wireless channels deteriorate the performance of the transport layer of the network [1]. Ensuring reliable end-to-end connections over versatile wired and wireless networks which become complex and heterogeneous, presents a demanding challenge in terms of its practical implementation. This challenge arose during the last decades as the subject of the researchers' interest [2]. One of the mechanisms that plays a critical role in TCP performance is the congestion control (CC) mechanism.

One of the main purposes of the CC mechanism is to probe the network for available capacity by dynamically increasing or decreasing the amount of data (also known as window size) that can be sent over a communication link. This approach enables avoiding the congestion of the network that can be caused by a large burst of data during transmission [3]. Node or link congestion in wired networks and blockage and beam misalignment in wireless networks can seriously affect the TCP CC mechanism. This further results in poor end-to-end transmission performance. Conventional TCP CC algorithms are not able to differentiate between the potential causes of packet loss. They can occur due to a lack of wired link capacity or poor wireless channel quality[4].

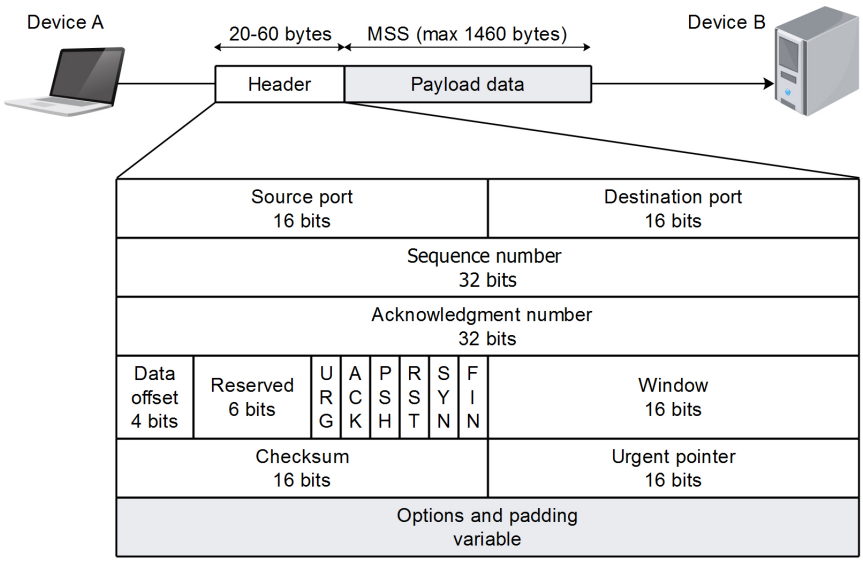


Figure 1. Structure of the TCP segment.

To achieve the specified performance and to accommodate the requirements of different communication systems, in the past, significant efforts have been invested by industry and academic researchers to reduce or eliminate the aforementioned shortcomings. Still, with every new generation or type of network, new challenges in terms of TCP CC arise. Integration of new types or generations of networks with the existing in terms of data CC, requests for new attempts dedicated to the optimization of CC mechanism. This is mostly done through the development of new TCP-based CC algorithms. In this topic review, an overview of the most popular single-flow and multi-flow TCP algorithms used for CC has been presented.

The rest of the paper is organized as follows. An overview of the TCP is given in Section 2. The TCP CC mechanism is explained in Section 3. A review of the popular single-flow and multi-flow CC algorithms is presented in Section 4. In Section 5, research challenges and future directions in possible improvements of traditional CC algorithms for implementation in complex heterogeneous networks are discussed. Some concluding remarks are given in Section 6.

2. Overview of Transmission Control Protocol

The TCP is a connection-oriented, reliable end-to-end protocol that guarantees the ordered delivery of byte streams in a full-duplex inter-process communication [5], [6]. The TCP is designed to complement the Internet Protocol (IP). This is a connectionless protocol and does not guarantee reliable data delivery. Initially, TCP was developed as a protocol to support military computer communication systems. Nowadays, TCP is de facto the standard for reliable end-to-end communication over the Internet. TCP and IP are the two main protocols on which the entire Internet communication resides and together with their supporting protocols, they form the Internet protocol suite known as TCP/IP.

The TCP provides reliable communication services in the transport layer of the Open System Interconnection (OSI) reference model. The protocol interfaces with the upper layer application processes and the lower layer IP [6]. The protocol data unit (PDU) of the transport layer using TCP is called a segment. The TCP PDU contains protocol-specific fields known as the TCP header and payload data (Figure 1). The maximum size of the TCP payload data is characterised as the maximum segment size (MSS). The fields of the TCP header specified in the TCP header format are [6]: source port, destination port, sequence number, acknowledgment number, data offset, reserved, control bits, window, checksum, urgent pointer, options, and padding (Figure 1). The TCP is responsible for establishing and terminating the connection, for the reliable communication between hosts, for the flow control and for the CC over unreliable networks.

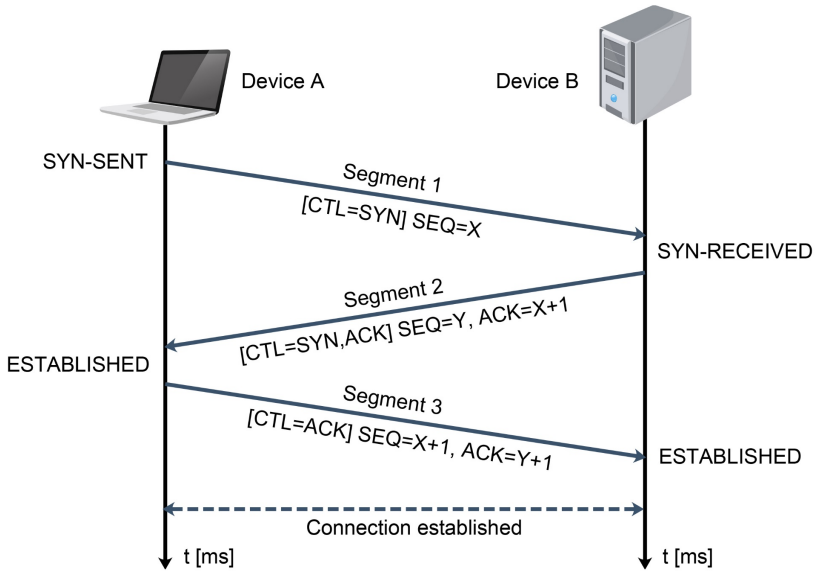


Figure 2. TCP three-way handshake process.

2.1. Establishing and Terminating a Connection

The process of establishing a TCP connection involves a three-way handshake procedure. A three-way handshake is a procedure where the initial connection parameters are agreed to exchange in three messages or segments between the sender and the receiver device (Figure 2). The first segment of the three-way handshake process is the SYN segment where device A sends the TCP segment to device B when they want to establish a reliable connection. In this segment, the TCP segment of device A indicates the initial sequence number that will be used for communication and the SYN control bit in the TCP header is set to 1 (Figure 2). The sequence number is a 32-bit random number maintained by both

sides of the TCP session to ensure that every byte of sent data can be acknowledged. This exchange of a sequence of numbers ensures the ordered delivery of the segments. After receiving the SYN segment, device B responds with the TCP segment by sending an acknowledgment (ACK) for the received SYN segment together with its initial sequence number as the second segment of the process (Figure 2). In this segment, the ACK control bit in the TCP header is set to 1, indicating that the acknowledgment field is valid. Finally, device A responds with the third TCP segment that acknowledges the SYN segment received from device B (Figure 2). Each side of the process acknowledges the sequence numbers with an ACK value that are one larger than the sequence number received, consequently acknowledging all earlier sequence numbers and identifying the next expected sequence number [7].

After completing the initial three-way handshake process, a full-duplex reliable communication is established and both ends can proceed with the application's process of data sending (Figure 2). Both ends of communication use sequence numbers in combination with acknowledgments to ensure that the ordered data delivery is completed. There are several variables that TCP maintains with respect of the send and received sequence numbers. These variables, among all of the other segment variables, are essential for maintaining a TCP connection and they are stored in a record called a transmission control block (TCB) [6].

TCP connection termination is a simple operation that allows the user who closes the connection to receive data until the other side also closes the connection [6]. The normal close sequence starts by sending a connection termination request as a FIN segment with a FIN control bit (flag) set to 1. When the other side involved in communication receives the FIN segment, it acknowledges its reception by sending an ACK segment followed by a FIN segment of its own. This segment indicates that the other side involved in the communication is ready to close the TCP connection. The initial side then receives the FIN segment and sends the ACK for the segment, allowing both ends to close their connection.

2.2. Flow Control

TCP uses a flow control mechanism to determine how much data the receiver is able to accept. The receiver uses a window field in the TCP header which represents the available buffer size of the receiver. This is in order to inform the sender about the maximum number of bytes that it is allowed to transmit [8]. The buffer size of the receiver depends on the speed with which the receiving application reads the data. The data stays on the receiving buffer until the receiving application reads the data. If the buffer memory of the receiver is full, the receiver advertises a window size of zero which informs the sender to stop sending more data. After the receiver retrieves the data from the buffer, it can advertise the new window size, allowing the sender to resume sending the data. The described mechanism is the generally accepted mechanism for flow control in IP networks.

3. TCP Congestion Control

3.1. Network Congestion and TCP CC Mechanism

Network congestion is the result of a network node being overwhelmed with more data than it can process. Buffers are added to the communication nodes to prevent packet loss caused by the burst of packets. This consequently increases the delay of each packet passing through the buffer, which results in the degradation of the network performance [9]. On the other hand, network nodes with large buffers can cause an excess buffering of the packets. This is known as a bufferbloat problem [10]. This consequently leads to long queuing delays and overall network throughput degradation. The maximum bandwidth of the network link is referred to as a bottleneck bandwidth and it is determined by the bandwidth of the fully saturated network link. In a scenario where the TCP data rate is less than the bottleneck bandwidth, no congestion occurs and the delivery rate corresponds to the sending rate [9]. As the sending rate exceeds the bottleneck bandwidth, the buffers start to fill to the point where the buffers are full, causing the network nodes to start dropping packets. The optimal operation point of TCP is the point where the sending rate is equal to the bottleneck bandwidth. The TCP CC is a mechanism that aims to prevent network congestion and to ensure efficient network utilisation while working near to the optimal operation point.

The TCP CC is one of the main parts of the TCP protocol. Over the years, it has experienced numerous improvements through the application of different algorithms. The first TCP specification considered only flow control as a mechanism to prevent the buffer overflowing on the receiver end, while neglecting the congestion of the network itself [6]. The general idea of TCP CC is to prevent the sender from overflowing the network by determining the available capacity. Congestion avoidance and control were first introduced in 1988 [11] after a series of "congestion collapses" occurred on the Internet. The standard TCP CC mechanism is based on the additive increase multiplicative decrease (AIMD) algorithm, which

incorporates four phases: slow start, congestion avoidance, fast retransmit and fast recovery[3]. An initial version of CC only requires the implementation of a slow start and the congestion avoidance phase. The fast retransmit and fast recovery phases were introduced later.

3.1.1. The Slow Start and Congestion Avoidance TCP Mechanism

The interdependence between the congestion window (*cwnd*) size and the transmission round-trip-time (RTT) of the TCP segment for the TCP version based on a slow start with congestion avoidance has been presented in Figure 3a. In this approach, the slow start and congestion avoidance phases are implemented using a stated *cwnd* variable that controls the amount of data that can be sent before receiving an ACK (Figure 3a). The *cwnd* size is a sender-specific variable that is maintained for each TCP session. The bandwidth-delay product (BDP), which determines the maximum amount of incoming data over a network link is considered to be the ideal value of the *cwnd* size.

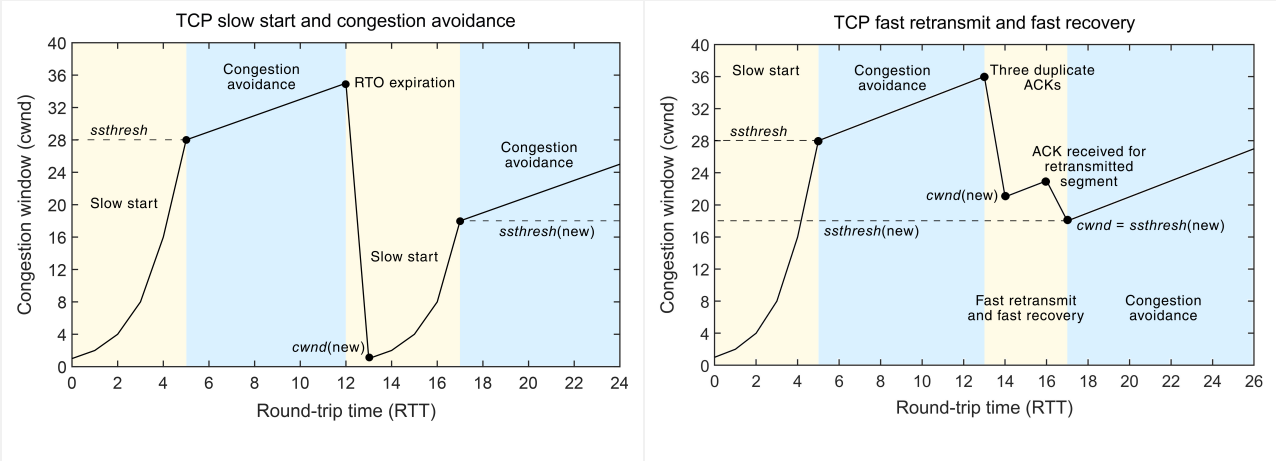


Figure 3. Standard working principles of the TCP CC mechanism related to the interdependence between *cwnd* and RTT for: (**a**) the TCP slow start and congestion avoidance mechanism; (**b**) the TCP fast retransmit and fast recovery mechanism.

The network congestion is evaluated based on the received ACK packets of transmitted data. The slow start phase is the initial phase in the beginning of transmission (Figure 3a). In this phase, the TCP quickly probes the available capacity while avoiding the network congestion that can occur in the beginning of the transmission. After every successful transmission of data and the reception of an appropriate ACK for said data, *cwnd* is increased and the sender can send more data (Figure 3a). After the three-way handshake process, the initial window (*iw*) is determined and can be large as 10 maximum segment size (MSS) [12]. In the slow start phase, the *cwnd* is increased by one MSS for each ACK received, doubling its size for every RTT. The *cwnd* will continue increasing the rate until either the *ssthresh* (slow-start threshold) is reached (Figure 3a), packet loss occurs or *cwnd* exceeds the *rwnd* (receiver window) size.

After *cwnd* exceeds the *ssthresh* size, the TCP CC mechanism enters the congestion avoidance phase (Figure 3a). The purpose of the congestion avoidance phase is to slowly probe the network for more available capacity, by increasing the *cwnd* less aggressively than in the slow start phase. The connection stays in the congestion avoidance phase until congestion is detected. If congestion is detected during the congestion avoidance phase using the retransmission time-out (RTO) expiration parameter, the TCP connection enters the slow start phase where the *ssthresh* is set to half of the current *cwnd* size and the *cwnd* is decreased to one MSS (Figure 3a). The process of adjusting the congestion window (*cwnd)* continues according to the described mechanism.

3.1.2. The Fast Retransmit and Fast Recovery TCP Mechanism

Waiting for the RTO to expire leads to a long period of connection inactivity. Because of that, a new mechanism called fast retransmit with fast recovery was introduced as a part of the TCP CC [12] (Figure 3b). The interdependence between the congestion window (*cwnd*) size and the transmission RTT of the TCP segment for the TCP version based on a fast retransmit with a fast recovery mechanism has been presented in Figure 3b. The general idea of a fast retransmit with a fast recovery is not to replace the RTO parameter, but to work in parallel, allowing the sender to retransmit the lost segment even if the timeout has not expired. In a TCP communication, when a segment arrives out of order, the receiver resends the ACK that was last sent, causing the sender to receive duplicate ACKs. This signals to the sender that the

segment is either lost or delayed. This causes a reordering of the segments. In the case of segment reordering, it is assumed that only one or two duplicate ACKs will be sent. The case when three ACKs are received by the sender before the RTO has expired indicates that the segment is lost.

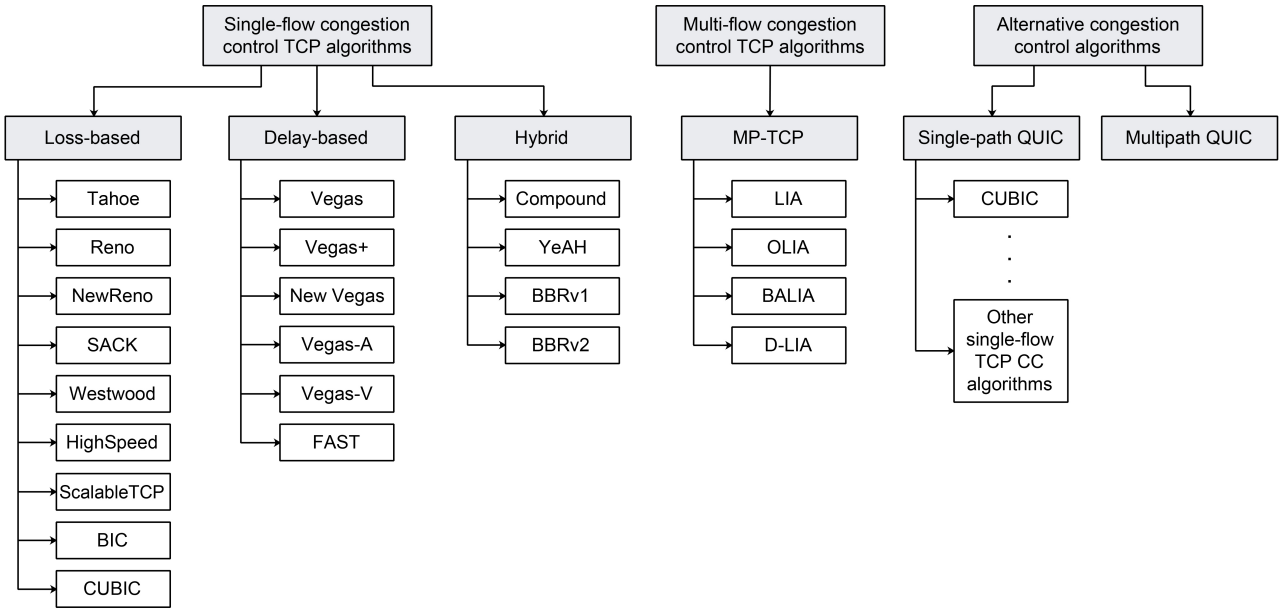


Figure 4. Main categories and types of the TCP CC algorithms.

This triggers a fast retransmit and a fast recovery mechanism (Figure 3b). In the standard implementation of the fast retransmit with a fast recovery phase, the sender, instead of going into the slow-start phase, decreases the *ssthresh* by one-half of the current *cwnd* (but by no less than two segments). Additionally, the sender retransmits the missing segment and sets the *cwnd* to a value equal to *ssthresh* plus 3 times the segment size [13] (Figure 3b). Each time, the sender receives an additional duplicate ACK after the third *cwnd* is incremented by MSS. When the ACK that acknowledges the new data arrives, *cwnd* is set to the new value of *ssthresh* (which is set at the moment of the initial triggering of the mechanism). It then enters the congestion avoidance phase, as shown in Figure 3b [13].

The aforementioned TCP CC mechanism is the standard implementation of the mechanism. It can vary depending on the CC algorithm used. There are numerous CC algorithms that have been developed and proposed as CC emerged as one of the most studied areas of internet research during the last few decades [14]. TCP CC, although not part of the original TCP implementation, has become an essential element of TCP as the entire TCP performance depends on the CC algorithm.

4. TCP CC Algorithms

CC algorithms are being developed with a focus on optimizing different metrics and making trade-offs between the various metrics in order to accommodate the different working environments and use cases. The main metrics that the Transport Modelling Research Group (TMRG) of the Internet Research Task Force proposed for evaluating CC algorithms were[15]: throughput, delay, packet loss rates, fairness, convergence times and robustness. Depending on the desired working environment, TCP CC algorithms are usually designed to make trade-offs between these metrics. There are three main categories of TCP CC algorithms as shown in Figure 4: (1) loss-based algorithms that use packet loss as an indicator of congestion, (2) delay-based algorithms that predict packet loss based on RTT measurements and (3) hybrid algorithms in which, the loss-based and delay-based methods are combined. Table 1 presents the comparison of the different CC algorithms according to their classification and year of introduction.

4.1. Loss-Based TCP CC Algorithms

A Tahoe TCP was the first CC algorithm that implemented a fast retransmit phase [11] followed by the Reno TCP [16] which included a fast recovery procedure (Table 1). The Tahoe TCP and Reno TCP are loss-based algorithms, and both consider RTO and duplicate ACKs as an indication of segment loss due to network congestion. The main difference between the two CC algorithms is in how they respond after receiving three duplicate ACKs and how they perform the fast retransmitting of the segments. After the fast retransmit phase, the Tahoe TCP switches to the slow start phase, whereas in contrast, the Reno TCP avoids the slow start phase by entering the fast recovery phase. In high-traffic environments where multiple segment losses occur in a single congestion window, the Reno TCP has shown a performance decrease as it can only detect single-segment losses [17].

Table 1. Overview of the most relevant single-flow TCP CC algorithms.

CC Algorithm	Introduction Year	Type	Features
Tahoe [11]	1988	Loss-based	Slow start, congestion avoidance and a fast retransmit mechanism.
Reno[16]	1990	Loss-based	Introduced fast recovery mechanism.
NewReno [18]	1999	Loss-based	Fast recovery modification to allow multiple retransmissions.
SACK [19]	1996	Loss-based	Selective ACK option.
Westwood [20]	2001	Loss-based	Introduced a faster recovery mechanism that controls the sending rate according to the available bandwidth estimation.
HighSpeed [21]	2003	Loss-based	Introduced a modified TCP response function to allow for a faster *cwnd* increase and a faster recovery time in situations with a high *cwnd* size.
Scalable TCP[22]	2003	Loss-based	After each received ACK during the RTT, algorithm increases the *cwnd* size in proportion with the defined constants. After the packet loss decreases *cwnd* by a smaller factor, the standard CC is exploited.
BIC[23]	2004	Loss-based	Uses the binary search increase and additive increase techniques to determine the *cwnd* size.
CUBIC [24]	2008	Loss-based	Uses the cubic function for *cwnd* control characterised by a steady state and a maximal probing behaviour.
Vegas[25]	1994	Delay-based	Modification of TCP Reno that predicts network congestion before an actual loss of segments occurs. Uses a fine-grained RTT estimation and has a very efficient segment retransmission schedule.
Vegas+ [26]	2000	Delay-based	Modification of TCP Vegas that introduced an aggressive mode to overcome fairness issues when competing with TCP Reno.
New Vegas [27]	2005	Delay-based	Implemented three server-side modifications of TCP Vegas to overcome performance issues in a high latency environment.
Vegas-A [28]	2005	Delay-based	Implemented modified congestion avoidance mechanism to address fairness, rerouting and bias against high bandwidth connections issues of TCP Vegas in wired and satellite networks.
Vegas-V [29]	2012	Delay-based	Modification of TCP Vegas-A that addresses fairness and aggression issues when competing with TCP Vegas, TCP Vegas-A and TCP Reno flows.

FAST [30]	2004	Delay-based	Designed for high-speed long-latency networks. Adjusts the *cwnd* according to the feedback information of the average RTT and average queuing delay. Uses scaling parameters to effectively utilise the network capacity.
Compound [31]	2006	Hybrid	Based on the loss-based slow start phase. During the congestion avoidance phase, it uses a combination of two components, a standard loss-based and a new scalable delay-based component.
YeAH [32]	2007	Hybrid	Performs a dynamic exchange between a slow mode during the congestion avoidance phase and a fast mode during the fast recovery phase.
BBRv1 [33]	2016	Hybrid	Builds an explicit model of the network using the estimated RTT and the estimated available bottleneck bandwidth in order to prevent congestion.
BBRv2 [34]	2018	Hybrid	Uses ECN signals, improved fairness with CUBIC and lower packet loss rates for the optimisation of the TCP CC performance.

TCP NewReno is an improved version of the TCP Reno algorithm. In order to overcome the performance issues of the TCP Reno, the TCP NewReno introduces a slight modification of the TCP Reno's fast recovery mechanism (Table 1) [18]. After receiving multiple duplicate packets, like its predecessor, the TCP NewReno enters the fast retransmit phase. However, the TCP NewReno does not exit the fast recovery phase until all of the outstanding data (send without received ACK) at the time of entering the fast recovery phase is acknowledged, thus preventing multiple *cwnd* reductions.

TCP selective acknowledgments (SACKs) have been introduced to overcome the limitations of the algorithms that use cumulative ACKs (Table 1). This approach allows for the retransmission of more than one segment per RTT. In an established TCP connection, the receiver uses the selective ACKs (SACKs) option to inform the sender about all successfully received segments, thus allowing the sender to retransmit only the missing segments in one RTT[19].

TCP Westwood is a CC algorithm that uses a server-side modification of the TCP Reno *cwnd* control mechanism (Table 1) [20]. TCP Westwood improves the performance of TCP Reno, especially in lossy wireless networks due to its robustness against sporadic wireless network errors. It uses a mechanism called faster recovery where instead of halving *cwnd* after three duplicate ACKs, the mechanism adjusts the *cwnd* and *ssthresh* parameters based on the end-to-end estimation of the available bandwidth. The bandwidth of the connection is estimated through the regular monitoring of the ACKs returning rate. After the congestion event, the size of the *cwnd* and *ssthresh* parameters is set according to the bandwidth estimate, thus ensuring a faster recovery as opposed to the TCP Reno response [20].

HighSpeed TCP is a modification of the classical TCP CC mechanism [3] that aims to address the limitations of connections with large congestion windows (Table 1) [21]. In the situation of a small *cwnd*, the operation of the HighSpeed TCP is the same as in the standard CC. On the other hand, when the current *cwnd* is greater than the specified parameter, the HighSpeed TCP uses a modified TCP response function that allows for a faster *cwnd* increase rate and a faster recovery time after packet loss. A variation of the HighSpeed TCP known as the Scalable TCP [22] is optimised for high-speed wide area networks. It is based on a simple modification of the traditional CC mechanism [3]. The main goal of the Scalable TCP algorithm is to update the *cwnd* in a scalable fashion. This means that the recovery times are proportional only to the RTT of the connection, which makes the algorithm more robust in high BDP networks.

A binary increase congestion control (BIC) TCP algorithm has been developed to address unfairness issues and to provide bandwidth scalability in high-speed networks with large delays (Table 1) [23]. The BIC TCP algorithm uses a binary search increase and additive increase as the two techniques used to determine the *cwnd* size. Binary search increase is a search technique involving aggressive initial bandwidth probing in a situation where there is a large difference between the current *cwnd* size and the target *cwnd* size. The mechanism becomes less aggressive when the current *cwnd* size gets closer to the target *cwnd* size. In the case of network congestion, the BIC TCP reduces *cwnd* from performing multiplicative decreases. After a significant *cwnd* decrease, the BIC algorithm performs an additive increase scheme by increasing *cwnd* linearly. The increase then becomes logarithmic [23]. By combining the binary search increase with the additive increase technique, the BIC algorithm ensures RTT fairness and faster convergence times.

The authors of [24] proposed a high-speed TCP variant called CUBIC to overcome the fairness and complexity issues of the previous CC algorithms (such as the TCP BIC). TCP CUBIC has become the dominant CC algorithm on the Internet [35] and it is currently the default algorithm distributed by Linux (Table 1).

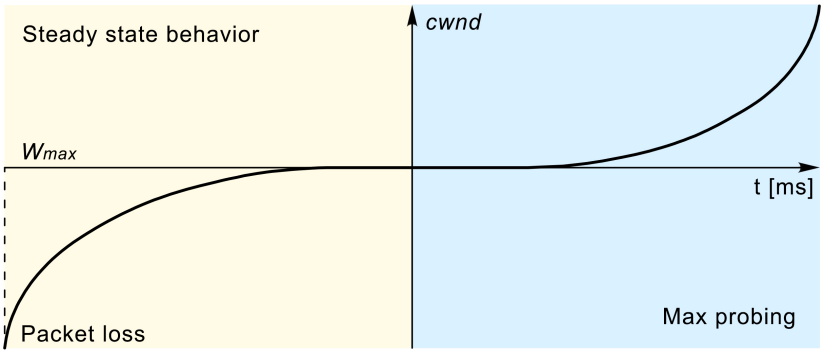


Figure 5. TCP CUBIC window growth function after the packet loss event.

The performance of the TCP CUBIC algorithm is visualised in Figure 5. To adjust the *cwnd* size, the TCP CUBIC uses a cubic function that is characterised by the aggressive growth of the window size following the event when a packet loss is detected. When the *cwnd* size approaches the point of the last congestion event (marked as in Figure 5), it slows its growth to almost zero. After reaching the point, the algorithm slowly probes the network by increasing the *cwnd* size at a slow rate. As it moves away from the , it rapidly accelerates its growth rate (Figure 5). Using a cubic function for calculating the *cwnd* growth rate ensures fairness among the concurrent flows and good performance properties [14].

4.2. Delay-Based TCP CC Algorithms

All aforementioned loss-based CC algorithms use segment loss as a congestion indicator and they are based on utilising a reactive congestion adaptation method. These algorithms are opposed to proactive delay-based CC algorithms (Figure 4) [14]. A typical representative of loss-based CC algorithms is TCP Vegas (Figure 4). This was introduced as a delay-based modification of the TCP Reno algorithm (Table 1). It predicts network congestion before actual segment loss occurs. TCP Vegas uses a fine-grained RTT estimation that increases the accuracy of the computed timeout period, which leads to a very efficient segment retransmission schedule (Table 1) [25]. The TCP Vegas uses a modified retransmission mechanism based on calculating the RTT estimation used for detecting segment loss before waiting for the reception of the three duplicate ACKs (Table 1). This approach increases the algorithm recovery time. It also uses a modified version of the slow start and congestion avoidance phases, which are thoroughly explained in Ref. [25]. The TCP Vegas delay-based approach can improve the overall TCP flow throughput by keeping the sending rate stable. However, due to the slow *cwnd* growth, this approach can suffer from performance issues in high-speed networks. In networks with concurrent loss-based flows, the TCP Vegas algorithm shows a decrease in performance due to the unfair resource allocation [14].

In the literature, several modifications of the TCP Vegas algorithm have been proposed. They are dedicated to addressing the limitations of bottlenecks during link sharing when many TCP flows are present, and to alleviate the performance issues in the case of links with high latency. A modified version of the TCP Vegas, called the TCP Vegas+ has been introduced to overcome fairness issues between the TCP Reno and the TCP Vegas (Figure 4) [26]. The algorithm introduced an aggressive mode to maintain fair throughput when competing with TCP Reno flows. To address performance issues of TCP Vegas over high latency links, a TCP New Vegas (Figure 4) has also been introduced. Compared with TCP Vegas, TCP New Vegas implements three sender-side modifications of TCP Vegas [27]. Furthermore, another modification of TCP Vegas has been developed, called TCP Vegas-A (Figure 4). The algorithm uses the modified congestion avoidance mechanism of TCP Vegas to address issues such as fairness against TCP New Reno flows, rerouting issues, bias against high bandwidth connections and fairness issues between old and new connections in wired and satellite networks [28]. The authors in Ref.[29] proposed a modified version of TCP Vegas-A, called TCP Vegas-V (Figure 4). This CC algorithm improves performance regarding fairness and aggression features in the network environment containing competing TCP Vegas, TCP Vegas-A and TCP Reno flows.

Another representative of delay-based CC algorithms is the FAST TCP algorithm that is designed for high-speed long-latency networks (Table 1) [30]. Based on the feedback information of the average RTT and average queuing delay, the FAST TCP periodically updates *cwnd* in order to control the packet transmission. This feedback information is provided by a latency estimation of each packet sent. Fairness and the number of buffered packets in the network are both controlled by a scaling parameter. FAST TCP adjusts its *cwnd* size constantly according to the number of buffered packets. If the number of buffered packets is far from the defined scaling parameter, the FAST TCP algorithm increases or decreases the

cwnd size, effectively utilising the network capacity. On the other hand, when the number of buffered packets reaches the target scaling parameter, the FAST TCP algorithm adjusts its *cwnd* by a small amount, thus ensuring the networks' stability [36].

4.3. Hybrid TCP CC Algorithms

Hybrid TCP CC algorithms are a combination of loss-based and delay-based algorithms (Figure 4). In a congested network situation or a network with high link utilisation having short bottleneck queues, hybrid algorithms tend to use a delay-based CC approach. On the other hand, in a high-speed network with a low link utilisation, hybrid algorithms tend to use a more aggressive CC approach that is characteristic of loss-based algorithms [9]. Some of the well-known representatives of hybrid CC algorithms are Compound TCP, Yet Another Highspeed (YeAH) TCP, and the TCP Bottleneck Bandwidth and Round-trip Propagation Time (BBR) algorithm (Figure 4).

The Compound TCP algorithm is representative of a hybrid-based CC algorithm developed for high-speed and long-distance networks. It was the default TCP algorithm in Microsoft Windows operating systems (Table 1) [31]. Compound TCP is designed to ensure high network utilisation, RTT and TCP fairness using a combination of loss-based and delay-based approaches. In the slow start phase, Compound TCP has the same aggressive behaviour characteristic as standard loss-based CC algorithms. During the congestion avoidance phase, it uses a combination of a standard loss-based component and a new scalable delay-based component. The delay-based component is based on the TCP Vegas algorithm, and it is controlled by a new state variable called *dwnd* (delay window). By encompassing the delay-based component, Compound TCP provides faster network utilisation and a reduction of the sending rate according to the bottleneck queue. This ensures better TCP fairness and the reduction of the sending rate in a packet loss event.

YeAH TCP is a hybrid variant of the high-speed TCP CC algorithm which uses two operation modes known as fast and slow modes respectively (Figure 4) [32]. During the fast recovery phase, the fast mode is initiated and the *cwnd* is incremented aggressively as in the Scalable TCP algorithm (Table 1). In the congestion avoidance phase, the slow mode is triggered by adjusting the *cwnd* size in the TCP Reno fashion. YeAH TCP uses the number of packets in the bottleneck queue determined through RTT measurements. This packet number is used as a parameter to determine the state of the algorithm. The main benefits of YeAH TCP are its high efficiency in situations where there are small link buffers typical of high BDP networks. An important benefit of the YeAH TCP algorithm compared to loss-based algorithms is its improved fairness.

The TCP BBR algorithm was developed by Google to overcome the problems faced by loss-based CC algorithms such as throughput issues in a small buffer scenario and the bufferbloat problem in a network with large bottleneck buffers (Table 1) [33]. It is a cutting-edge CC algorithm that creates an explicit CC model based on the recent measurements of the RTT and the delivery rate of the network. Using an explicit CC data model, the algorithm adjusts the sending rate accordingly. According to the periodic estimation of the available bandwidth and minimal RTT, the BBR algorithm prevents congestion by ensuring low delay and high throughput operations. The sending rate is determined through the pacing gain process, which is the main mechanism that the BBR uses to control the sending behaviour. This is done by maximising the rate at which the BBR schedules packets. When the bottleneck bandwidth (*BtlBw*) has been estimated, the BBR deliberately reduces the pacing gain to drain the queues in order to estimate the round trip propagation time (*RTprop*). Periodical measurements of BDP calculated as the multiplication of the *BtlBw* and *RTprop* parameters ensure performance within the Kleinrock optimal operating point [37], where data delivery rate is maximised and delay is minimised[38]. However, the initial version of the BBR has drawbacks which are mainly reflected in unfairness when competing with loss-based CC, including a large number of packet losses and increased queuing delays [39]. For that reason, an improved version known as a BBRv2 has been introduced to alleviate these problems [34]. Google has deployed TCP BBR across all of its services, which has resulted in a higher throughput, reduced latency and better overall connection quality. Even though TCP BBR is a relatively new CC algorithm, [35] showed that after TCP CUBIC, it is the second most dominant CC algorithm used on the Internet. It is reasonable to predict that TCP BBR will surpass TCP CUBIC in the near future.

4.4. TCP Fairness

In a situation of multiple concurrent TCP flows competing for the same bandwidth on the network link, some TCP CC algorithms may receive more of a bandwidth share than other TCP flows. The challenge of satisfying bandwidth allocation fairness is a serious problem in TCP CC. Hence, one of the main goals that CC algorithms need to accomplish is to ensure fairness among flows that are competing for the same bottleneck bandwidth [40]. Due to the competition among different TCP flows, fair bandwidth allocation among TCP flows can severely degrade the performance of the CC algorithms characterised by a less aggressive approach in the competition for bandwidth.

Loss-based CC algorithms (e.g., TCP Reno) have an aggressive nature as packet loss is the only indicator of congestion. This nature is mainly reflected in the continuous increase of *cwnd* until packet loss occurs. On the other hand, delay-based algorithms (e.g., TCP Vegas) use RTT estimation to predict network congestion before an actual loss has occurred, reducing the sending rate accordingly. In a network with many competing flows that are loss-based, delay-based flows cannot get their fair throughput share as shown in Ref. [41]. This is because flows with an aggressive loss-based approach will obtain a larger amount of bandwidth than their fair share. Likewise, the fairness issue has been detected between the most dominant CC algorithms on the Internet, and the CUBIC and BBR algorithms are no exception. It has been shown that TCP BBR favours small buffers and gets a significantly higher share when competing with CUBIC. In contrast, when using large buffers, BBR cannot compete with CUBIC because it occupies most of the bottleneck bandwidth as shown in Ref. [39]. When deploying CC algorithms in dynamic environments, such as the mmWave 5G network where a large number of lines of line-of-sight (LOS) and non-line-of-sight (NLOS) transitions are expected, fairness issues should be seriously considered. The NLOS state of the TCP flow causes an increase in RTT, which consequently results in bandwidth unfairness among the flows [42].

4.5. TCP Optimisation Techniques

The purpose of TCP CC mechanisms is to address congestion on the network and to fully utilise the network resources. Besides CC, several TCP optimisation techniques have been developed to enhance the operations of CC. To solve the excessive buffering of packets on the network and thus preventing bufferbloat, active queue management (AQM) techniques are proposed as opposed to a basic drop-tail queuing mechanism. AQM is a technique based on proactively discarding packets from the buffer before the queue is full, thus reducing the risk of a queuing delay, preventing the bufferbloat problem and proactively reducing network congestion[43]. AQM schemes are deployed on the network device buffers (e.g., routers, BSs, etc.), rather than as part of a TCP implementation. The benefit of using the AQM is to maintain a small queue size to prevent overflowing the buffers with a large burst of packets. Furthermore, keeping the queue size small reduces the queuing delay and minimises the overall end-to-end delay of the network. Finally, the AQM avoids the impact of global TCP synchronisation and this contributes to the increased throughput and better utilisation of the network [44].

Random early detection (RED) is an example of a basic AQM algorithm that addresses global synchronization problem, minimise network congestion and limits network delay[45]. Due to the configuration difficulties and performance issues regarding bursty traffic of RED, a new technique called controlled delay (CoDel) has been developed. The CoDel is an AQM technique [46] that treats differently low delay queues and queues that continuously buffer packets causing the increased delay. CoDel operates by regularly monitoring the minimum queuing delay in specific intervals and by discarding packets when the minimum queue delay is exceeded. By using queue delay instead of buffer queue occupancy as a queue management metric, CoDel improves network utilization, which further contributes to achieving high throughput and better queue management [47].

Another TCP optimisation method that is used in combination with CC algorithms is the explicit congestion notification (ECN) technique. The ECN is a congestion detection technique based on the setting of the codepoint (mark) in the ECN field of the IP packet header [48]. Based on this codepoint mark annotation, the ECN-capable device can signal on the transport layer regarding the incipient congestion before actual congestion occurs. This allows the CC algorithms to adjust their *cwnd* size accordingly. ECN is generally used in combination with AQM techniques, where ECN marks the IP packet header based on the information obtained from the AQM scheme. Enabling the ECN technique can result in packet loss reduction, queuing delay reduction, and improved throughput in the connection [48]. Instead of dropping packets, an AQM technique may interact with ECN to mark packets and therefore indicate congestion prior to the actual packet loss occurs[49].

Since regular ECN informs the sender only once per RTT about incipient congestion, a more accurate ECN feedback scheme (AccECN) has been proposed. It is based on allowing more than one feedback signal to be transmitted per RTT [50]. Although AccECN was proposed in 2011, it is still an Internet-draft working document expected to be standardised.

4.6. Multistream TCP Variants and Alternatives

Although TCP is the dominant transport protocol used today, there are emerging variants and alternatives that can improve overall network performance (Figure 4). The two most relevant protocols are multipath TCP (MP-TCP) and Quick UDP Internet Connection (QUIC). These protocols have been widely used and recently standardized by an Internet Engineering Task Force (IETF). MP-TCP has been introduced as a TCP variant for multipath data communication and QUIC is a UDP-based TCP alternative initially developed for hypertext transfer protocol (HTTP) traffic (Figure 4).

4.6.1. Multipath TCP

Standardized by an IETF in 2020, multipath TCP (MP-TCP) is an extension to the TCP that handles multiple paths simultaneously for a single data stream [51]. Multipath connection is envisioned to utilize the presence of multiple network interface cards (NICs) commonly found in today's devices (e.g., smartphones). Hence, MP-TCP can use multiple network interfaces (e.g., 4G, 5G, Wi-Fi, Ethernet, etc.) to create multiple subflows that use multiple paths for a single connection. Using a multipath approach for end-to-end communication can improve utilization of network resources, enhance throughput, and ensure more robust and resilient communication [51].

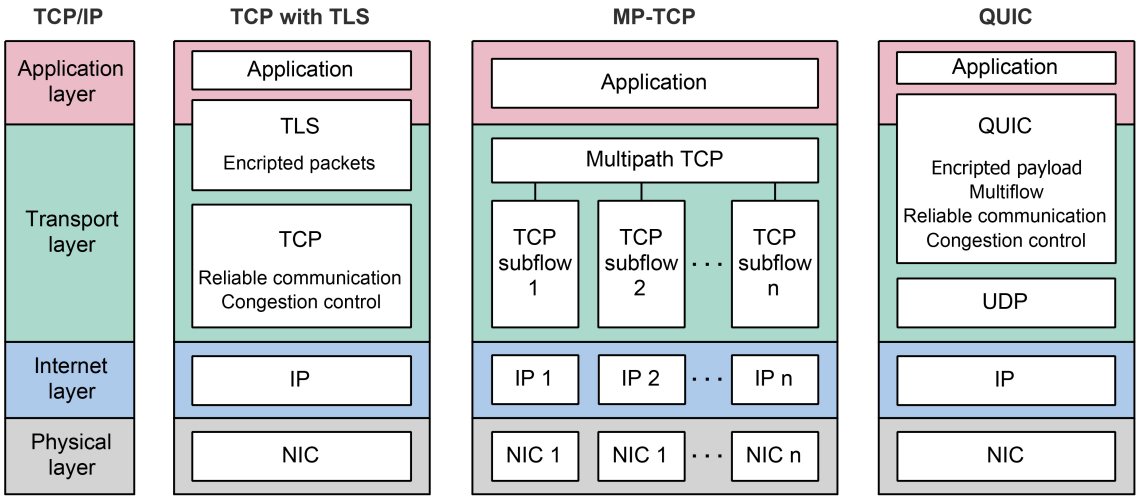


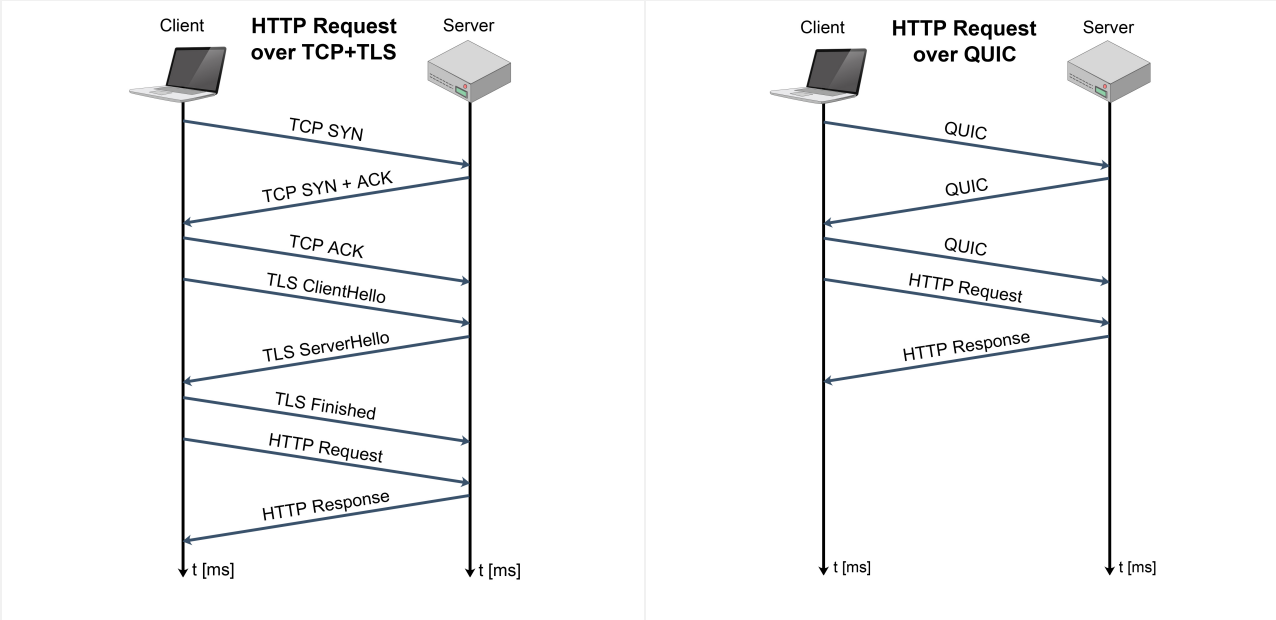
Figure 6. Comparison between TCP with TLS, QUIC and MP-TCP protocol stacks.

In Figure 6, a comparison between the TCP protocol stack and the MP-TCP protocol stack has been presented. It can be seen that MP-TCP divides a single TCP connection into multiple different TCP flows. This dividing requires a CC algorithm that can control the transmission rate for each subflow. To control the data transmission rate for each subflow, MP-TCP CC must satisfy three design goals related to ensuring fair bottleneck bandwidth usage and robustness of the connection [52]. First, multiple subflows of a single multipath connection should perform at least as well as a single path connection would on the best path available. Second, multipath connection should not use more capacity than a single path TCP connection. Finally, an MP-TCP should balance congestion in such a way as to move data towards paths that are less congested.

Depending on the *cwnd* control method for each subflow, CC mechanisms used with MP-TCP are characterized as uncoupled or coupled. The uncoupled CC mechanism handles each subflow independently allowing each subflow to have its own instance of TCP CC algorithm. On the other hand, the coupled CC mechanism manages *cwnd* in a coupled manner, by considering the characteristics of all other subflows. The initial coupled CC algorithm (Figure 4), called the linked increases algorithm (LIA) defined in Ref.[52], suffers from performance issues. As demonstrated in Ref. [53], LIA transmits a large amount of data over paths that are congested and can have aggressive behavior with respect to a legacy single-path TCP. To address these issues (Figure 4), the authors in Ref. [53] have proposed the opportunistic linked increases algorithm (OLIA). Additionally, authors in Ref. [54] (Figure 4), proposed a balanced linked adaptation algorithm (BALIA). The performance of BALIA is based on balancing responsiveness, TCP friendliness and *cwnd* oscillations. However, these algorithms are based on the legacy TCP Reno CC algorithm and follow the AIMD scheme, and as such, they cannot satisfy previously presented three design goals for operation in a 5G mmWave environment [55] Since the aforementioned coupled CC algorithms are primarily focused on the increase of the *cwnd,* while neglecting the *cwnd* decreasing mechanism [56] (Figure 4), authors in Ref. [57] proposed a loss-based MP-TCP CC algorithm called Dynamic-LIA (D-LIA). Instead of halving the *cwnd* after packet lost occurrence, the D-LIA decreases the *cwnd* by a dynamically determined factor that depends on the interval between each packet loss[57]. Using this approach, *cwnd* can reach its optimal size much faster than the regular the TCP AIMD mechanism and thus reduce overall network latency and better utilize network resources. Although the D-LIA achieves better overall performance in terms of throughput and fairness, the authors have detected a downside related to increased packet retransmissions [57].

4.6.2. QUIC Protocol

The quick UDP Internet connections (QUIC) protocol is initially developed by Google as an alternative to TCP (Figure 4). It uses user datagram protocol (UDP) at the transport layer (Figure 6).



(**a**) (**b**)

Figure 7. Comparison of connection establishment and message exchange for: (**a**) TCP with TLS, (**b**) QUIC protocol.

Although standardized in May 2021 by IETF, initial implementation and deployment started in 2012 and QUIC today represents the authenticated and encrypted by default Internet transport protocol with the tendency of eventually replacing TCP and Transport Layer Security (TLS) protocols on the web [58] (Figure 6). QUIC is designed to overcome CC issues of transport and application layer for web-based applications. An upcoming third version of the hypertext transfer protocol (HTTP/3) is designed with QUIC as a built-in transport layer protocol [59] and all major web browsers are starting to support it.

According to Figure 6, QUIC includes the TLS layer with its own framing. This ensures permanent authentication and encryption of the connection and makes the initial connection establishment faster. The handshake messages exchange for TCP with TLS and QUIC have been presented in Figure 7. It can be seen that the QUIC handshake only requires one round-trip between client and server to complete, while joint TCP with TLS handshakes requires two round-trips.

The main advantages of QUIC over TCP include: 0-RTT connection establishment which significantly reduces latency, improves CC with loss detection and minimize head-of-line-blocking delay by supporting the multiplexed operation. Furthermore, QUIC has introduced a pluggable CC interface that provides more flexibility over TCP. Namely, QUIC uses generic CC signaling that can support different CC algorithms, thus allowing flexible algorithm selection[60]. Additionally, QUIC uses monotonically increasing packet numbers to ensure proper packet order. This ensures avoiding retransmission ambiguities and simplified loss detection. It also includes information about the delay between the receipt of a packet and the acknowledgment (ACK) sent for that packet, thus allowing for a more precise RTT estimation. To reduce the packet losses caused by packet bursts, QUIC uses a packet pacing mechanism that evenly spaces packet transmissions over time. It has been shown in Refs. [61], [62] that a packet pacing mechanism minimizes the probability of packet losses and supports data stream multiplexing. This eliminates head-of-line-blocking problems what can be especially beneficial over lossy wireless environments (e.g., 4G, 5G networks).

Motivated by the development of the MP-TCP protocol, authors in Ref. [63] have proposed a multipath QUIC (MP-QUIC) protocol (Figure 4). The main capability of MP-QUIC is the possibility to pool resources for a connection that uses multiple paths and to improve resistance to connection failures. This is especially important for today's multi-homed devices (e.g., smartphones) that need to be able to make an uninterrupted switch between different network interfaces. Multipath extension for QUIC (MP-QUIC) was introduced in 2017 and is described in detail in the IETF Internet draft document, which is currently in the process of standardization [64].

5. Research Challenges and Future Directions

It has been shown that there is no unique TCP CC algorithm that can satisfy all use cases and applications, especially in highly variable network environments (such as 5G [65] and future generations of cellular networks or Internet of Everything (IoE) applications). Developing an efficient transport layer CC algorithms that are able to effectively utilise the bandwidth and overcome the issues such as blockages, misalignment and handover in wireless networks and link or node

congestions (bottlenecks) in wired networks represents a great challenge. This is because the level of the *sstresh* (in Fig. 3a) is not constant and varies with the total network load. Hence, the CC problem is dominantly an optimization problem in which each traffic source must continuously adapt its data traffic transmission based on backward information received from the network.

Future solutions to optimal CC in complex heterogeneous networks composed of wireless, wired and a combination of wireless and wired parts will be in the development of an optimal solution for a particular situation. For example, in emerging usage scenarios where one device requires a high bandwidth priority and the other requires ultra-low latencies, the use of different TCP CC algorithms is expected. Still, many questions must be answered in the development of new CC algorithms such as: what is an optimal incremental or decremental policy of transmission rate for a specific application, should the CC be deployed on a hop-by-hop or end-to-end basis, by what means the network can signalize to the communicating points about congestion and what shud be prioritized, the low end-to-end latency or high link utilization?

Another issue related to CC algorithms is dedicated to ensuring fairness between connections sharing the same link and in defining a quantification of fairness. Scenarios in which multiple different TCP flows are controlled using different CC algorithms and compete for their fair share of the link capacity can pose a serious challenge in terms of practical realisation. More specifically, the problem of fairness for different TCP flows has not yet been successfully solved. Some approaches propose solving the fairness issues for critical applications (such as autonomous driving or telesurgery) where ultra-high reliability and ultra-low latency are expected, through the usage of only one TCP CC algorithm in an isolated environment.

Another challenge is to find a solution that will utilise the full wired link capacity or wireless channel bandwidth and minimise latency at the same time. Queuing delays as a consequence of large bottlenecks at the buffer level must be addressed as their impact can seriously deteriorate network performance. One of the promising solutions is the cross-layer approach where the transport layer of the OSI model can use the information obtained from the different layers to adjust the congestion window (*cwnd*) size accordingly.

Regardless of the method used for congestion detection, the main challenges to TCP CC that must be addressed in future networks are

- Latency vs. throughput trade-off challenge: There are various solutions to achieve low latencies. However, many of them are at the expense of bandwidth. It will be a challenge to find a solution that will achieve the best compromise for the growing number of different communication use cases and practical scenarios.
- Challenge of queuing delay and bufferbloat problem: The problem of excess packet buffering creating long queuing delays known as bufferbloat, was detected in many research studies. Nevertheless, it is a research area that still needs to be further explored.
- Optimization of energy consumption challenge: It must be considered in the development of CC algorithms. This is especially important for the wireless sensor networks and mobile devices, which operational activity depends on the duration of the battery power supply.
- The continuous and large increase of Internet Protocol data traffic challenge: For example, this is especially reflected in the emergence of new multimedia applications. Also, the advent of new types of sensor networks like Body Area Sensor Networks (BASN), Underwater Sensor Networks (UWSNs) or Wireless Multimedia Sensor Networks (WMSNs), impose new challenges in the development of adequate CC algorithms.

Although CUBIC is the dominant CC algorithm for the broad Internet traffic today, the BBR algorithm increased its share in terms of the practical implementation and it can be expected to become the dominant algorithm in the future. As the BBR algorithm is slowly replacing the CUBIC algorithm, further research regarding their mutual interaction is needed to ensure the stability of the Internet. Despite the fact that BBR achieves good performance in terms of maximising the throughput and minimising latency, in highly variable network environments with a massive number of connected devices, achieving an optimal network performance will be challenging.

Academic and industry researchers are constantly making efforts to improve traditional rule-based algorithms, that use predefined heuristics to address new requirements. On the other hand, machine learning (ML) TCP CC is in its early stages and it remains to be seen whether the future will be consistent with the traditional CC paradigm or if the future of TCP CC lies in intelligent ML algorithms. The deployment of the ML in TCP CC is still in its infancy and it is too early to expect any significant application at a higher level. However, with the advent of sixth-generation (6G) networks, ML will need to be considered as the dominant direction in the field of network CC. There are possible directions in the implementation of the ML for CC. The first one is based on the development of entirely new ML-based algorithms that will

completely replace the existing ones. The second is based on the integration of some of the ML properties into the existing algorithms and maintaining its backward compatibility with rule-based algorithms. The future will show which of these directions will dominate.

6. Conclusions

The implementation of new services, network generations and network types in wired and wireless networks, brings new challenges in the realisation of data traffic CC. In this encyclopedic work, the TCP and the basics of the CC mechanism have been presented. Also, a thorough survey of the most popular TCP algorithms used for CC is performed. An overview of surveyed TCP algorithms includes single-flow CC algorithms, multi-flow CC algorithms and alternative algorithms that have been envisioned as the future replacement for TCP.

Due to the challenges and limitations that conventional CC algorithms have, it is reasonable to expect further advancements of the TCP CC algorithms in the years to come. A discussion related to the main research challenges that must be addressed for the efficient implementation of CC algorithms in the upcoming networks has been presented. Also, the potential future directions related to the development and implementation of ML-based algorithms as the most promising candidates for the implementation of CC in complex future networks are elaborated.

Abbreviations

The following abbreviations are used:

4G	4th Generation Mobile Network
5G	5th Generation Mobile Network
6G	6th Generation Mobile Network
AccECN	Accurate Explicit Congestion Notification
ACK	Acknowledgment
BALIA	Balanced Linked Adaptation Algorithm
BBR	Bottleneck Bandwidth and Round-Trip Propagation Time
BDP	Bandwidth-Delay Product
BIC	Binary Increase Congestion Control
BS	Base Station
BtlBw	Bottleneck Bandwidth
CC	Congestion Control
CoDel	Controlled Delay
CTL	Control field
cwnd	Congestion Window

D-LIA	Dynamic Linked Increases Algorithm
ECN	Explicit Congestion Notification
eMBB	Enhanced Mobile Broadband
IETF	Internet Engineering Task Force
IoE	Internet of Everything
IP	Internet Protocol
IW	Initial Window
LIA	Linked Increases Algorithm
LOS	Line-of-Sight
MAC	Media Access Control
ML	Machine Learning
mmWave	Millimeter-Wave
MP-TCP	Multipath TCP
MP-QUIC	Multipath QUIC
ms	millisecond
MSS	Maximum Segment Size
NIC	Network Interface Card
NLOS	Non-Line-of-Sight
OLIA	Opportunistic Linked Increases Algorithm
OSI	Open System Interconnection
PUH	Push field
QUIC	Quick User Datagram Protocol Internet Connection
RST	Reset field
RTO	Retransmission Time-Out

RtProp	Round Trip Propagation Time
RTT	Round-Trip-Time
rwnd	Receiver Window
SACK	Selective Acknowledgment
SEQ	Sequence field
ssthresh	Slow-Start Threshold
SYN	Synchronize (field)
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URG	Urgent field
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
YEAH	Yet Another Highspeed Transmission Control Protocol

References

1. Mateo, P.J.; Fiandrino, C.; Widmer, J. Analysis of TCP Performance in 5G mm-Wave Mobile Networks. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; p p. 1–7.
2. Zhang, M.; Polese, M.; Mezzavilla, M.; Zhu, J.; Rangan, S.; Panwar, S.; Zorzi, A.M. Will TCP Work in mmWave 5G Cell ular Networks? IEEE Commun. Mag. 2019, 57, 65–71.
3. Allman, M.; Paxson, V.; Blanton, E. TCP Congestion Control. RFC 5681. September 2009. Available online: https://tool s.ietf.org/html/rfc5681 (accessed on 31 April 2021).
4. Poorzare, R.; Aug, A.C. Challenges on the Way of Implementing TCP Over 5G Network. IEEE Access 2020, 8, 176393 –176415.
5. Cerf, V.G.; Kahn, R.E. A Protocol for Packet Network Intercommunication. IEEE Trans. Commun. 1974, 22, 637–648.
6. Postel, J. Transmission Control Protocol. RFC 793. September 1981. Available online: https://tools.ietf.org/html/rfc793 (ac-cessed on 31 April 2021).
7. Peterson, L.L.; Davie, B.S. Computer Networks: A Systems Approach; Elsevier, 2012. Available online: https://book.syst emsapproach.org (accessed on 31 April 2021).
8. Clark, D.D. Window and Acknowledgement Strategy in TCP. RFC 813. July 1982. Available online: https://tools.ietf.org/ html/rfc813 (accessed on 31 April 2021).

9. Turkovic, B.; Kuipers, F.A.; Uhlig, S. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. ar Xiv 2019, arXiv:1903.03852.
10. Gettys, J.; Nichols, K. Bufferbloat: Dark Buffers in the Internet: Networks without effective AQM may again be vulnerabl e to congestion collapse. Queue 2011, 9, 40–54.
11. Jacobson, V. Congestion Avoidance and Control. Comput. Commun. Rev. 1988, 18, 314–329.
12. Chu, J.; Dukkipati, N.; Cheng, Y.; Mathis, M. Increasing TCP's Initial Window. RFC 6928. April 2013. Available online: ht tps://tools.ietf.org/html/rfc6928 (accessed on 31 April 2021).
13. Stevens, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001. Janu ary 1997. Available online: https://tools.ietf.org/html/rfc2001 (accessed on 31 April 2021).
14. Afanasyev, A.; Tilley, N.; Reiher, P.; Kleinrock, L. Host-to-Host Congestion Control for TCP. IEEE Commun. Surv. Tutor. 2010, 12, 304–342.
15. Floyd, S. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166. March 2008. Available online: http s://tools.ietf.org/html/rfc5166 (accessed on 31 April 2021).
16. Jacobson, V. Modified TCP Congestion Avoidance Algorithm; Technical Report. 1990. Available online: ftp://ftp.ee.lbl.go v/email/vanj.90apr30.txt (accessed on 31 April 2021).
17. Mascolo, S.; Casetti, C.; Gerla, M.; Sanadidi, M.Y.; Wang, R. TCP westwood: Bandwidth estimation for enhanced trans port over wireless links. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networki ng (MobiCom'01), Rome Italy, 16–21 July 2001; pp. 287–297.
18. Mathis, M.; Mahdavi, J.; Floyd, S.; Romanow, A. TCP Selective Acknowledgment Options. RFC 2018. October 1996. Av aila-ble online: https://tools.ietf.org/html/rfc2018 (accessed on 31 April 2021).
19. Floyd, S. HighSpeed TCP for Large Congestion Windows. RFC 3649. December 2003. Available online: https://tools.iet f.org/html/rfc3649 (accessed on 31 April 2021).
20. Kelly, T. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. ACM SIGCOMM Comput. Commun. Rev. 2003, 32, 83–91.
21. Xu, L.; Harfoush, K.; Rhee, I. Binary increase congestion control (BIC) for fast long-distance networks. In Proceedings of the IEEE INFOCOM, Hong Kong, China, 7–11 March 2004.
22. Ha, S.; Rhee, I.; Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. ACM SIGOPS Oper. Syst. Rev. 2008, 4 2, 64–74.
23. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. AC M SIGCOMM Comput. Commun. Rev. 1994, 24, 24–35.
24. Hasegawa, G.; Kurata, K.; Murata, A. Analysis and Improvement of Fairness between TCP Reno and Vegas for Deploy ment of TCP Vegas to the Internet. In Proceedings of the 2000 International Conference on Network Protocols, Osaka, Japan, 14–17 November 2000; pp. 177–186.
25. Sing, J.; Soh, B. TCP New Vegas: Improving the Performance of TCP Vegas over High Latency Links. In Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications, Cambridge, MA, USA, 27–29 Jul y 2005; pp. 73–82.
26. Srijith, K.; Jacob, L.; Ananda, A. TCP Vegas-A: Improving the Performance of TCP Vegas. Comput. Commun. 2005, 28, 429–440.
27. Zhou, W.; Xing, W.; Wang, Y.; Zhang, J. TCP Vegas-V: Improving the performance of TCP Vegas. In Proceedings of the International Conference on Automatic Control and Artificial Intelligence (ACAI 2012), Xiamen, China, 3–5 March 2012; pp. 2034–2039.
28. DWei, X.; Jin, C.; Low, S.H.; Hegde, S. FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Tran s. Netw. 2006, 14, 1246–1259.
29. Tan, K.; Song, J.; Zhang, Q.; Sridharan, M. A Compound TCP Approach for High-speed and Long Distance Networks. I n Proceedings of the IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, Barc elona, Spain, 23–29 April 2006; pp. 1–12.
30. Baiocchi, A.; Castellani, A.P.; Vacirca, F. Yeah-tcp: Yet another highspeed TCP. In Proceedings of the 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, CA, USA, 7–9 February 2007.
31. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. Commu n. ACM 2017, 60, 58–66.

32. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Swett, I.; Vasiliev, V.; Jha, P.; Seung, Y.; Mathis, M.; Jacobson, V. Bbrv2: A mod-el-based congestion control. In Proceedings of the IETF 102th Meeting, Montreal, QC, Canada, 14–20 July 2018; pp. 1–36.
33. Henderson, T.; Floyd, S.; Gurtov, A.; Nishida, Y. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582. April 2012. Available online: https://tools.ietf.org/html/rfc6582 (accessed on 31 April 2021).
34. Mishra, A.; Sun, X.; Jain, A.; Pande, S.; Joshi, R.; Leong, B. The Great Internet TCP Congestion Control Census. Proc. ACM Meas. Anal. Comput. Syst. 2019, 3, 1–24.
35. Mishra, A.; Sun, X.; Jain, A.; Pande, S.; Joshi, R.; Leong, B. The Great Internet TCP Congestion Control Census. Proc. ACM Meas. Anal. Comput. Syst. 2019, 3, 1–24.
36. Jin, C.; Wei, D.; Low, S.H.; Bunn, J.; Choe, H.D.; Doylle, J.C.; Newman, H.; Ravot, S.; Singh, S.; Paganini, F.; et al. FA ST TCP: From theory to experiments. IEEE Netw. 2005, 19, 4–11.
37. Kleinrock, L. Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications. In Proceedings of the International Conference on Communications (ICC '79), Berlin, Germany, 17–20 September 1979; Volume 3, pp. 43.1.1–43.1.10.
38. Scholz, D.; Jaeger, B.; Schwaighofer, L.; Raumer, D.; Geyer, F.; Carle, G. Towards a Deeper Understanding of TCP BB R Con-gestion Control. Available online: https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/IFIP-Networking-2018-TCP-BBR.pdf (accessed on 31 April 2021).
39. Hock, M.; Bless, R.; Zitterbart, M. Experimental Evaluation of BBR Congestion Control. In Proceedings of the IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, 10–13 October 2017.
40. Floyd, S. Congestion Control Principles. RFC 2914. September 2000. Available online: https://datatracker.ietf.org/doc/html/rfc2914 (accessed on 31 April 2021).
41. Mo, J.; La, R.J.; Anantharam, V.; Walrand, J. Analysis and comparison of TCP Reno and Vegas. In Proceedings of the IEEE INFOCOM'99, Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, New York, NY, USA, 21–25 March 1999.
42. Pieska, M.; Kassler, A.J.; Lundqvist, H.; Cai, T. Improving TCP Fairness over Latency Controlled 5G mmWave Communica-tion Links. WSA 2018. In Proceedings of the 22nd International ITG Workshop on Smart Antennas, Bochum, Deutschland, 14–16 March 2021.
43. Gong, Y.; Rossi, D.; Testa, C.; Valenti, S.; Täht, M.D. Fighting the bufferbloat: On the coexistence of AQM and low priority congestion control. In Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, 14–19 April 2013
44. Gomez, C.A.; Wang, X.; Shami, A. Intelligent Active Queue Management Using Explicit Congestion Notification. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 28–30 October 2019.
45. Floyd, S.; Jacobson, V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw. 1993, 1, 397–413.
46. Nichols, K.; Jacobson, V.; McGregor, A.; Iyengar, J. Controlled Delay Active Queue Management. RFC 8289. January 2018. Available online: https://tools.ietf.org/html/rfc8289 (accessed on 31 April 2021).
47. Al-Saadi, R.; Armitage, G.; But, J.; Branch, P. A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms. IEEE Commun. Surv. Tutor. 2019, 21, 3609–3638.
48. Fairhurst, G.; Welzl, M. The Benefits of Using Explicit Congestion Notification (ECN). RFC 8087. March 2017. Available online: https://tools.ietf.org/html/rfc8087 (accessed on 31 April 2021).
49. Baker, F.; Fairhurst, G. IETF Recommendations Regarding Active Queue Management. RFC 7567. July 2015. Available online: https://datatracker.ietf.org/doc/html/rfc7567#section-2.2 (accessed on 31 April 2021).
50. Briscoe, B.; Kuehlewind, M.; Scheffenegger, R. More Accurate ECN Feedback in TCP. IETF, Internet-Draft. February 2021. Available online: https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn (accessed on 31 April 2021).
51. Ford, A.; Raiciu, C.; Handley, M.; Bonaventure, O.; Paasch, C. TCP Extensions for Multipath Operation with Multiple Ad-dresses. RFC 8684. March 2020. Available online: https://datatracker.ietf.org/doc/html/rfc8684 (accessed on 31 April 2021).
52. Raiciu, C.; Handly, M.; Wischik, D. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356. October 2011. Available online: https://datatracker.ietf.org/doc/html/rfc6356#section-3 (accessed on 31 April 2021).
53. Khalili, R.; Gast, N.; Popovic, M.; Boudec, J.L. MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution. IEEE/ACM Trans. Netw. 2013, 21, 1651–1665.

54. Peng, Q.; Walid, A.; Hwang, J.; Low, S.H. Multipath TCP: Analysis, Design, and Implementation. IEEE/ACM Trans. Netw. 2016, 24, 596–609.
55. Polese, M.; Jana, R.; Zorzi, M. TCP in 5G mmWave networks: Link level retransmissions and MP-TCP. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1 May 2017.
56. Fu, F.; Zhou, X.; Dreibholz, T.; Wang, K.; Zhou, F.; Gan, Q. Performance comparison of congestion control strategies for multi-path TCP in the NORNET testbed. In Proceedings of the 2015 IEEE/CIC International Conference on Communications in China (ICCC), Shenzhen, China, 2–4 November 2015.
57. Lubna, T.; Mahmud, I.; Cho, Y.-Z. D-LIA: Dynamic congestion control algorithm for MPTCP. ICT Express 2020, 6, 263–268.
58. Iyengar, J.; Thomson, M. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. May 2021. Available online: https://datatracker.ietf.org/doc/html/rfc9000 (accessed on 31 April 2021).
59. Bishop, M. Hypertext Transfer Protocol Version 3 (HTTP/3). IETF, Internet-Draft. February 2021. Available online: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34 (accessed on 31 April 2021).
60. Iyengar, J.; Swett, I. QUIC Loss Detection and Congestion Control. RFC 9002. May 2021. Available online: https://datatracker.ietf.org/doc/html/rfc9002 (accessed on 31 April 2021).
61. Cook, S.; Mathieu, B.; Truong, P.; Hamchaoui, I. QUIC: Better for what and for whom? In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017.
62. Yu, Y.; Xu, M.; Yang, Y. When QUIC meets TCP: An experimental study. In Proceedings of the 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), San Diego, CA, USA, 10–12 December 2017.
63. Coninck, Q.D.; Bonaventure, O. Multipath QUIC: Design and Evaluation. In Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17), Incheon, Korea, 12–15 December 2017.
64. Coninck, Q.D.; Bonaventure, O. Multipath Extensions for QUIC (MP-QUIC). IETF Internet-Draft. May 2021. Available online: https://datatracker.ietf.org/doc/draft-deconinck-quic-multipath/07/ (accessed on 31 April 2021).
65. Lorincz, J.; Klarin, Z.; Ožegović, J. A Comprehensive Overview of TCP Congestion Control in 5G Networks: Research Challenges and Future Perspectives. Sensors 2021, 21, 4510, 1-41.