Congestion Control Schemes for Multipath QUIC

Julian Gassner, Daniel Petri*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: julian.gassner@tum.de, petriroc@net.in.tum.de

***Abstract*—The current IETF draft for MPQUIC employs a per path instance of QUIC's default congestion control algorithm for calculating path congestion windows. The lack of fairness between paths and the negative performance impact of competing standard single-path protocols are notable problems. Building upon the lessons learned from MPTCP, two novel approaches to MPQUIC congestion control called CC-OLIA and S2B2C mitigate these issues.**

***Index Terms*—MPQUIC, MPTCP, congestion control algorithm, OLIA, CC-OLIA, S2B2C**

1. Introduction

The Multipath Extension for QUIC (MPQUIC) [1] started development in October 2017 and is currently in its 13th IETF [2] draft iteration [1]. It seeks to standardize the use of multiple QUIC paths simultaneously within a connection to improve throughput and reliability [1]. However, to date, the draft still recommends the use of one instance per path of the standard QUIC congestion control algorithm as defined in RFC 9002 [3]. This causes several problems, most notably the lack of fairness between paths [1] and the negative performance impact on competing standard single-path protocols [4]. This paper looks at how TCP's Multipath Extension (MPTCP) [5] works and how it handles congestion control (CC). We discuss the basics of MPQUIC and analyze its default CC algorithm, and show two novel approaches to MPQUIC CC proposed by H. Wang et al. and Deng et al. named CC-OLIA and S2B2C that mitigate these issues.

2. Background

In this Chapter, we look at TCP's Multipath Extension [5] and its main CC algorithms LIA [6] and OLIA [7]. In addition, we analyze the functionality of the QUIC Multipath Extension [1]. We contextualize this to lay the groundwork for discussion of the multipath QUIC CC algorithms.

2.1. TCP's Multipath Extension

According to RFC 8684 [5], Multipath TCP (MPTCP) allows multiple paths to be used simultaneously between peers. This behavior contrasts standard TCP connections where only one path is used at a time. In this context, a path is defined by TCP's 4-tuple containing the source address, the source port, the destination address, and the destination port of the connection. Using multiple TCP paths has the advantage of making better use of available network resources. It also results in higher throughput and improved resilience to network failures. The connection multiplexing takes place entirely within the TCP layer, and MPTCP connections provide the same interfaces to the other layers as standard TCP connections. This makes it possible to utilize the benefits of MPTCP without the need to change intermediaries and the applications that work upon it. [5]

2.1.1. MPTCP Functionality. The multiple path connections are established using distinct network interfaces with different IP addresses on both hosts. From an external perspective, a MPTCP connection behaves and operates like any other TCP connection, the network layer is divided into multiple *subflows*. A sub-flow represents a stream of segments over a single path from a given TCP connection. The MPTCP extension creates, manages, and deletes these subflows, which all operate on a path identified by the 4-tuple. The number of currently active subflows can vary throughout the connection. [5]

Each sub-flow is an instance of a classic TCP flow with the addition of a new TCP option type. This option type has various subtypes used during the different states of a connection. The initial connection establishment for a MPTCP connection follows the same flow as normal TCP connections (SYN, SYN/ACK, ACK) and utilizes only a single path. However, each packet contains a MP_CAPABLE subtype option to declare that the sender of that packet wants to utilize MPTCP for the given connection, which MPTCP version (`v0` or `v1`) the sender is capable of running, and various flags that define the connection's used features. It also exchanges the keys of the connection partners used to authenticate later-created subflows. [5]

After the connection is established, hosts can create any number of new subflows using currently unused IP address pairs. Hosts can indicate available IP addresses on their side of the connection using the ADD_ADDR option subtype. As with the master connection, each new subflow starts with the standard TCP connection establishment procedure. However the subflows' packets do not contain the MP_CAPABLE subtype option but the MP_JOIN subtype option. It enables the flow establisher to inform the receiver which connection this subflow belongs to by sending a token generated from the receiver's key exchanged during the master connection establishment, which identifies the master connection. This happens in the SYN segment. The SYN, SYN/ACK, and ACK seg-

 doi: 10.2313/NET-2025-11-3_05

ments all contain the MP_JOIN option subtype and use this option subtype, besides the already mentioned purpose, for replay attack and integrity protection. Once the connection setup is finished, MPTCP is ready to transfer data. Data transfers over the different subflows happen by splitting the data to be sent and reassembling it on the receivers' side. This is done by using the DSS option subtype. It carries a Data Sequence Number (DSN), which enumerates the separated data used to indicate how to reassemble it. The DSS also contains a Data ACK field used to acknowledge given data parts by their DSN in their given subflow. This ACK is one of two available ACKs in MPTCP, with the other one being the standard TCP ACK used to acknowledge the segments in general and not a specific DSN. [5]

2.1.2. Congestion Control Algorithms for MPTCP. Several types of CC algorithms are used for MPTCP. RFC 8684 [5] suggests using the Coupled CC algorithm, also known as the Linked Increases Algorithm (LIA), which is defined in RFC 6356 [6]. The idea of LIA is to keep most of the original TCP CC algorithms, such as the slow start, fast recovery, and fast retransmit algorithms defined in RFC 5281 [8], and only change the congestion avoidance algorithm [6]. For this RFC 6356 [6] defines several variables similar to the variables defined in RFC 5281 [8] for single path connections:

- $cwnd_i$: Congestion windows of the i[th] subflow in bytes.
- $cwnd_{\text{total}}$: Sum of all congestion windows across all subflows in bytes.
- p_i: Loss rate of a subflow i.
- rtt_i: Round trip time of a subflow i.
- MSS_i: Maximum segment size of a subflow i in bytes.

Each time a segment ACK is received in a subflow during the congestion avoidance phase as defined in RFC 5281 [8], the variable $cwnd_i$ is updated using Formula 1, where b is the number of bytes acknowledged in the ACK and the aggressiveness α is defined in Formula 2 [6].

$$\Delta cwnd_i = \min\left(\frac{\alpha \cdot b \cdot MSS_i}{cwnd_{\text{total}}}, \frac{b \cdot MSS_i}{cwnd_i}\right) \tag{1}$$

$$\alpha = cwnd_{\text{total}} \cdot \frac{\max_i\left(\frac{cwnd_i}{rtt_i^2}\right)}{\left(\sum_i \frac{cwnd_i}{rtt_i}\right)^2} \tag{2}$$

Using this approach, LIA achieves two goals for multipath CC algorithms. The first goal, called "Improve Throughput" [6] states that the multipath approach performs at least as well as a single path approach does on its best path. This is done by introducing α in Formula 2 to ensure that the combined throughput of all subflows aligns with what one single TCP flow would achieve. Secondly, the multipath approach can only put the same load on any given resource, for example, an intermediate router, on any given path as a single path approach would ("Do no harm" [6]). LIA implements this in the calculation of the $cwnd_i$ increase in Formula 1 to ensure that the increase never exceeds a standard TCP flow congestion window increase. However, the RFC also points out a third desirable goal for multipath CC algorithms, which cannot be achieved by using LIA. It states that a multipath CC algorithm must be able to offload as much traffic as possible from congested paths with respect to goals one and two ("Balance congestion" [6]). Furthermore, LIA shows so-called *flappiness*, meaning that when all subflows exhibit the same congestion behavior, LIA tends to assign the entire total congestion window to one subflow while all other subflows have a congestion window of 0. [6]

To mitigate these issues, the Opportunistic Linked-Increases CC algorithm for MPTCP (OLIA) [7] is developed [7]. OLIA introduces 4 new relevant variables, alongside the already known variables from LIA [7]:

- l_r: $\max\{l_{1r}, l_{2r}\}$, where l_{1r} denotes the amount of acknowledged bytes between the last two packet loss events, and l_{2r} denotes the amount of acknowledged bytes since the last loss event.
- best_paths: Set of paths that maximize the ratio $\frac{l_r^2}{RTT_r}$.
- max_cwnd_paths: Subset of paths with the biggest $cwnd_i$ out of best_paths.
- collected_paths: All paths out of best paths which are not in max_cwnd_paths.
- all_paths: Set of all available paths.

Using this variables OLIA calculates the congestion window per flow during the congestion avoidance phase. Similar to LIA, the other TCP congestion algorithms remain untouched and α_r defines the aggressiveness in the $cwnd_i$ increases, however OLIA uses different α_r formulas depending on whether the path is in best_path, max_cwnd_paths or collected_paths as can be seen in Formula 4 [7]. The Formula used to calculate the increase of $cwnd_i$ can be seen in Formula 3 [7] where b is the number of acknowledged bytes. [7]

$$\Delta cwnd_i = \left(\frac{cwnd_i}{rtt_i^2} \Bigg/ \left(\sum_{p \in \text{all\_paths}} \frac{cwnd_i}{rtt_i}\right)^2 + \frac{\alpha_r}{w_r}\right) \cdot MSS_i \cdot \text{b} \tag{3}$$

$$\alpha_r = \begin{cases} \frac{1}{|\text{collected\_paths}|} & \text{if } r \in \text{collected\_paths}, \\ -\frac{1}{|\text{max\_w\_paths}|} & \text{if } r \in \text{max\_w\_paths} \\ & \wedge\ |\text{collected\_paths}| > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Using this methodology, OLIA can achieve the two goals already achieved by LIA, as well as goals 3, making this algorithm superior to LIA [7]. Furthermore, it fixes the flappiness of LIA [7]. However, as Walid et al. [9] find, OLIA sometimes does not react adequately when it comes to condition changes along the network paths under certain conditions.

2.2. Multipath Extension for QUIC

The Multipath Extension for QUIC [1] defines how the QUIC transport protocol can use multiple paths similar to MPTCP [1].

2.2.1. Functionality of MPQUIC. The Multipath Extension for QUIC follows the same idea as TCP's Multipath Extension. It takes an existing transport layer protocol and extends it to use multiple network paths simultaneously by adapting and extending already existing QUIC functionalities. Although QUIC already supports path migration, it does not have the capability to use multiple paths simultaneously. MPQUIC extends this capability to support concurrent path usage. [1]

In MPQUIC, paths are identified by a path ID. Similar to MPTCP, MPQUIC uses a new transport parameter to negotiate the use of MPQUIC in the initial client-server handshake. This transport parameter is called initial_max_path_id, and its presence signals that an endpoint is capable of using MPQUIC. The value of the parameter defines the maximum number of paths. The initiator sends it in the initial client crypto frame and the server also appends it to its initial crypto frame if it supports it. If the multipath handshake is successful, the endpoints start using PATH_ACK frames instead of ACK frames. [1]

PATH_ACK frames are an extension of QUIC's standard ACK frames, which are used to acknowledge packets in the context of a given path using the path IDs. The creation of a new path works by sending a PATH_NEW_CONNECTION_ID frame to a connection peer. It must contain a new connection ID and a linked new path ID used to identify the new path. The connection peer responds with its own PATH_NEW_CONNECTION_ID frame containing the same path ID and its own connection ID. [1]

After the path initiation, a path is validated with a PATH_CHALLENGE and PATH_RESPONSE frame sent by both connection partners to make sure that the path can be used. During the following transmission of data packets, the connection IDs are used to identify the path. Reassembling based on the stream offsets remains functionally the same as in normal QUIC operation, regardless of the paths used, as one stream can take multiple paths simultaneously. [1]

3. Congestion Control for MPQUIC

This Chapter discusses the default CC algorithm used by MPQUIC and present two novel approaches to MPQUIC CC proposed by H. Wang et al. [4] and Deng et al. [10] named CC-OLIA and S2B2C that improve various aspects of the default CC algorithm.

3.1. Default Congestion Control Algorithm

The Multipath Extension for QUIC specification draft [1] suggests utilizing a per-path instance of the CC algorithm used by the QUIC Transport protocol.

3.1.1. Functionality. MPQUIC maintains a congestion window per path, which limits how much data can be in flight at any given time. The slow start threshold known from TCP is initially set to infinity [11]. This threshold defines up to which congestion window size the slow start phase is used. As the slow start threshold is set to infinity, QUIC initially operates in a slow start phase. During the slow start phase, the congestion window, which is recommended to be set to 10 times the maximum datagram size, is increased by the bytes acknowledged in the PATH_ACK frames. This makes the congestion window grow exponentially. This behavior equals the behavior of TCP's slow start phase [11]. Once a loss occurs, MPQUIC enters a recovery phase. A loss is detected either by a missing acknowledgment or when a probe timeout (PTO) occurs. The PTO is started when a segment is sent on a specific path and is calculated as defined in Formula 5, where $smoothed\_rtt$ is the estimated RTT on the network path as defined in the MPQUIC specification, $4 \cdot rttvar$ is the variation of $srtt$, $gran$ specifies the time granularity and $maxdel$ is the maximum delay that can occur on the receiver side before sending the acknowledgment. [1] [3] [12]

$$\text{PTO} = \text{srtt} + \max\left(4 \cdot \text{var}, gran\right) + \text{max\_del} \quad (5)$$

After the recovery phase is entered the slow start threshold is set to the congestion window divided by 2. The congestion window must then be abruptly or slowly reduced to the threshold during the recovery phase. If further losses occur during the recovery phase no further reduction of the congestion window is performed. Once an acknowledgment from a packet sent during the recovery phase is received, the recovery phase is left, and the congestion avoidance phase is entered. During the congestion avoidance phase, an Additive Increase Multiplicative Decrease is used to increase the congestion window similar to TCP's congestion avoidance phase [11]. Should a packet loss, as defined before, occur during the congestion avoidance phase, the algorithm again enters the recovery phase. If only packet loss occurs during an implementation-specific timeframe, persistent congestion is assumed, and the congestion window is reset. [1] [3] [12]

3.1.2. Discussion. Referring back to the goals of multipath transport protocols defined in RFC 6356 by Raiciu et al. [6] and discussed by us in Section 2.1.2, one finds that this algorithm does not achieve the "Do no harm" goal as it actively competes with other paths for overlapping resources [4]. Wang et al. [4] find that this can significantly harm single-path applications. According to RFC 6356 [6] the issue can be fixed by adapting the principles of LIA to MPQUIC, however no concrete implementation is specified.

3.2. Modified Congestion Control Algorithms

Due to the reasons mentioned in Section 3.1.2, there is a need to modify the default MPQUIC CC mechanisms such that they fulfill the desirable goals for multipath CC algorithms.

3.2.1. Modified Congestion Control Algorithms using OLIA. H. Wang et al. [4] present a modified algorithm that adapts the ideas of OLIA to MPQUIC in the context of mobile networks called CC-OLIA. CC-OLIA is quite similar to OLIA [7], but instead of only modifying the congestion avoidance phase, they also modify the slow start phase according to a coupled slow start algorithm initially proposed by Y. Wang et al. [13] for MPTCP.

This slow start algorithm introduces an aggressiveness factor similar to what LIA and OLIA use to limit the total

congestion window of an entire MPTCP connection for MPTCP. This makes sure that the slow start phase, with its initial exponential growth across the paths, does not instantly congest shared parts of the network resulting in faster communication speeds for MPTCP. [13]

H. Wang et al. [4] adapt this slow start algorithm to work with the similar slow start mechanisms of each path of MPQUIC and use it as long as no packet loss occurs during the slow start phase. If packet loss occurs during the slow start phase, a multiplicative decrease mechanism to reduce the congestion window. This principle is also used by TCP in its slow start phase [11]. Afterward, the algorithm switches to the congestion avoidance phase where the congestion window of each path i is incremented according to $\Delta cwnd_i$ of OLIA as described in Section 2.1.2. Should a packet loss occur during the congestion avoidance phase CC-OLIA switches into the *Packet Loss Classification* algorithm. [4]

This algorithm tries to determine if the packet loss occurred randomly or was part of a congestion by checking how many events occurred in a defined interval. The $last\_cutback$ variable defines this interval. It stores the latest packet number used when the last decrease of the congestion window occurred. Should packet loss occur, the algorithm checks if the packet number of the lost packet is greater than $last\_cutback$, indicating whether the loss occurred randomly or was the start of a larger network degradation. If the former applies, the algorithm classifies the loss as Random Packet Loss (RPL) and leaves the congestion window unchanged but updates $last\_cutback$ to the current packet number used for new packets. [4]

Furthermore, the slowstart threshold is updated to the current congestion window. Should the packet number be smaller or equal to $last\_cutback$, the algorithm assumes continuous congestion, reduces the congestion window across all paths, and classifies the loss as Congestion Packet Loss (CPL). This behavior can also be seen in Formula 10 [4] where n denotes the amount of paths currently used and d represents the multiplicative decrease factor. All other variables are defined in Section 2.1.2. [4]

$$cwnd_i = \begin{cases} cwnd_i, & \text{if RPL,} \\ cwnd_i \left(\frac{n-1+d}{n}\right), & \text{if CPL.} \end{cases} \tag{10}$$

When evaluating CC-OLIA H. Wang et al. [4] find that they can reduce the transfer times of a 5 MB, 25 MB, and 35 MB file by 6.3%, 14.9%, and 16.2% respectively compared to an OLIA MPTCP implementation in the case of a shared bottleneck scenario. In the case of a non-shared bottleneck scenario they could reduce the transfer times for the same files by 18.7%, 29.8%, and 36.3%.

3.2.2. BBR based congestion control for MPQUIC. The Bottleneck Bandwidth and Round-trip propagation time algorithm (BBR) is a CC algorithm implemented for TCP and QUIC, which relies on measuring connection speed, round trip time, and packet loss to avoid congestions. It works by estimating a bandwidth-delay product (BDP) through probing to estimate the optimal throughput along the used network path, which does not overwhelm the bottleneck of that given path. [14]

BBR has four algorithmic phases: Startup, Drain, ProbeBW, and ProbeRTT. During the first phase, the Startup phase, BBR quickly increases the used network bandwidth until the bottleneck buffer on the network path fills up and packet loss or the delivery rate plateaus. This is used to find the maximum bandwidth bw and the minimum RTT RTT_{min}. The BDP is estimated by multiplying RTT_{min} with bw. The algorithm then moves on to the drain phase, where it quickly drains the intermediate buffers by ramping down the throughput until the data in flight equals the estimated BDP. BBR then periodically probes for more bandwidth and minimum RTT in order to adapt the BDP and the data in flight. [14]

While we could theoretically apply the principles of BBR to each individual path in MPQUIC, we would face the same problems as one would when using the default CC algorithm as defined in the MPQUIC draft [10]. Therefore, Deng et al. [10] developed a novel BBR-based CC algorithm called S2B2C, which utilizes the principles of BBR but ensures fairness. They do so by identifying MPQUIC paths that share a bottleneck and fairly adapt their pacing gain G during the ProbeBW phase. The pacing gain alongside other variables influences the rate at which data is sent and, therefore, the amount of data in flight [15]. Every time BBR tries to estimate the available bandwidth, it sets the pacing gain $\mathcal{G}$ to $\mathcal{G} \in [1.25, 0.75, 1, 1, 1, 1, 1, 1]$ in an 8-step process during the ProbeBW phase [10].

During the first step, the sending rate is set to 1.25 multiplied by bw. As this is done per path, Deng et al. assume that all paths S_1 where the RTT increases after this first step must share at least one bottleneck. The same goes for all paths S_2 during the second step of ProbeBW, where the pacing gain is set to 0.75. Therefore, $S_1 \cap S_2$ denotes all paths that share a bottleneck. Deng et al. then repeat this procedure for three entire 8-step runs to ensure that any other network noise does not influence the groupings. [10]

Afterwards, they do the same again for ProbeRTT as during ProbeRTT, the maximum amount of in-flight data is limited to 4 maximum segment sizes resulting in a significant increase in round trip times in other paths that share the same bottleneck. The set of these paths is called S_3. Calculating $S_1 \cap S_2 \cap S_3$ per path now yields a very probable set of paths that share the same bottleneck. For all paths $r \in \xi$ that share the same bottleneck G_r is set to

$$\mathcal{G}_r \in [1.25, 0.75, \alpha_r, \alpha_r, \alpha_r, \alpha_r, \alpha_r, \alpha_r]$$

where α_r is calculated according to Formula 6 [10] where S denotes the set of all available paths. During performance testing, Deng et al. find that this strategy ensures bottleneck fairness and balanced congestion. [10]

$$\alpha_r = 4 \left(\frac{bw_r \times \max_{r \in S}\{bw_r\}}{\sum_{r \in \xi} bw_r} - 1 \right) / 3 \tag{6}$$

4. Conclusion and Future Work

Over the course of this paper, we first looked into how MPTCP works and analyzed a non-exhaustive list of common CC algorithms used alongside MPTCP. After that, we dived into the core functionality of MPQUIC

and explained how its default CC mechanism works. We then discussed its shortcomings and introduced the novel adaptation CC-OLIA, which is based on MPTCP's OLIA CC algorithm, but has been adapted to work with MPQUIC. We also introduced S2B2C, a CC algorithm based on BBR, which uses an entirely different method to handle congestion compared to OLIA's more traditional approach.

Both algorithms show promising initial results and can overcome the shortcomings of MPQUIC's default CC algorithm. While this is a non-exhaustive list of CC algorithms for MPQUIC, we find that it represents the two most prominent groups of either BDP or LIA-based CC algorithms. An example of another implementation of a BDP-based MPQUIC CC algorithm is MACO [16], which was developed to provide congestion control for fast-moving MPQUIC-based satellite networks [16]. Regarding future work, S2B2C needs to be updated to support BBRv2 [17]. Furthermore, we find that there is a research gap when it comes to a CC algorithm for MPQUIC based on the Balanced Linked Adaptation CC Algorithm for MPTCP (BALIA) [9], a further development of OLIA, which overcomes its mentioned weaknesses [9].

References

[1] Y. Liu, Y. Ma, Q. D. Coninck, O. Bonaventure, C. Huitema, and M. Kühlewind, "Multipath Extension for QUIC," Internet Engineering Task Force, Internet Draft draft-ietf-quic-multipath-13, Mar. 2025, num Pages: 40. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/13

[2] "IETF," Mar. 2025. [Online]. Available: https://www.ietf.org/

[3] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9002

[4] H. Wang, Y. Liu, Z. Li, Y. Zhang, W. Gong, T. Jiang, T. Bi, and J. Zhou, "Cc-olia: A dynamic congestion control algorithm for multipath quic in mobile networks," *Digital Communications and Networks*, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864824001640

[5] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses," Internet Engineering Task Force, Request for Comments RFC 8684, Mar. 2020, num Pages: 68. [Online]. Available: https://datatracker.ietf.org/doc/rfc8684

[6] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," Internet Engineering Task Force, Request for Comments RFC 6356, Oct. 2011, num Pages: 12. [Online]. Available: https://datatracker.ietf.org/doc/rfc6356

[7] R. Khalili, N. Gast, M. Popovic, and J.-Y. L. Boudec, "Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP," Internet Engineering Task Force, Internet Draft draft-khalili-mptcp-congestion-control-05, Jul. 2014, num Pages: 11. [Online]. Available: https://datatracker.ietf.org/doc/draft-khalili-mptcp-congestion-control-05

[8] E. Blanton, V. Paxson, and M. Allman, "TCP Congestion Control," Internet Engineering Task Force, Request for Comments RFC 5681, Sep. 2009, num Pages: 18. [Online]. Available: https://datatracker.ietf.org/doc/rfc5681

[9] A. Walid, Q. Peng, J. Hwang, and S. H. Low, "Balanced Linked Adaptation Congestion Control Algorithm for MPTCP," Internet Engineering Task Force, Internet Draft draft-walid-mptcp-congestion-control-04, Jan. 2016, num Pages: 11. [Online]. Available: https://datatracker.ietf.org/doc/draft-walid-mptcp-congestion-control-04

[10] Z. Deng, Y. Liu, J. Liu, A. Argyriou, and D. Liu, "Bbr-based and fairness-guaranteed congestion control and packet scheduling for mpquic over heterogeneous networks," *Computer Communications*, vol. 224, pp. 213–224, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366424002160

[11] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[12] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[13] Y. Wang, K. Xue, H. Yue, J. Han, Q. Xu, and P. Hong, "Coupled slow-start: Improving the efficiency and friendliness of mptcp's slow-start," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[14] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, p. 58–66, Jan. 2017. [Online]. Available: https://doi.org/10.1145/3009824

[15] N. Cardwell, Y. Cheng, S. H. Yeganeh, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, Jul. 2017, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/00/

[16] W. Yang, L. Cai, S. Shu, and J. Pan, "Mobility-aware congestion control for multipath quic in integrated terrestrial satellite networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11 620–11 634, 2024.

[17] N. Cardwell, I. Swett, and J. Beshay, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-ietf-ccwg-bbr-02, Feb. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-ccwg-bbr/02/