Transport Protocols Analysis

A technical paper prepared for presentation at SCTE TechExpo24

Rahil Gandotra
Lead Architect
CableLabs
r.gandotra@cablelabs.com

Arun Yerra
Principal Mobile Network Architect
CableLabs
a.yerra@cablelabs.com

Table of Contents

List of Figures

1. Introduction

Future networks (fixed and mobile) are gearing up for demanding applications like immersive XR, self-driving cars, and healthcare robots. These applications are expected to demand more from the network in terms of QoS characteristics. In particular, such applications require low latency/jitter, high data rates, and highly reliable and available networks. Packets not delivered within the required latency/jitter budget will be wasted and the user experience will be significantly impacted.

The transport layer, operating between the network and application layers, is the first layer in the stack that functions on an end-to-end basis between the two communicating hosts. User experience and overall network performance depend heavily on how applications, the transport layer, and the network work in synergy. Transport protocols provide several critical functions to enable data exchange between applications on a network: process-to-process delivery, multiplexing and demultiplexing, flow control, congestion control, etc. The increasing heterogeneity of the network deployment scenarios and the diverse and challenging QoS requirements make the role of transport protocols more crucial and more complex to design.

The adoption of new transport-layer solutions is restricted due to several factors, and the research community is forced to work around these limitations and design innovative approaches to improve network performance. The widespread use of middleboxes, which often block unknown protocols or unrecognized extensions to known protocols, invalidates the end-to-end principle, thereby impeding the deployment of alternative protocols, leading to transport protocols ossification [1]. Furthermore, most operating systems implement transport functionalities (e.g., TCP and UDP) within the kernel space, exposing socket APIs to the applications, making the deployment of new solutions difficult and limiting the interfacing options between applications and the transport protocols. This has essentially led to most of the Internet traffic either using TCP, for applications demanding reliable delivery, or UDP, for applications preferring timeliness to reliability.

This paper focuses on two directions in transport layer research – alternate transport protocols, and multi-path approaches – that have materialized to solve the aforementioned problems. Alternate transport protocols, such as Datagram Congestion Control Protocol (DCCP), Stream Control Transmission Protocol (SCTP) and QUIC, were developed as alternatives to the legacy TCP and UDP protocols, aiming to solve some of their inherent issues in addressing specific application requirements. Multi-path protocols improve single-path protocols' (e.g., TCP and QUIC) throughput and resilience by leveraging multiple network paths. The 5G feature Access Traffic Steering, Switching and Splitting (ATSSS) specified by 3GPP employs these multi-path transport protocols to utilize both the 3GPP access (e.g., 5G New Radio (NR)) and the non-3GPP access (e.g., Wi-Fi) to provide improved performance.

The rest of the paper is organized as follows: in Section 2, we highlight the main issues present in TCP and UDP and describe how the alternate protocols are designed to overcome them. Section 3 provides an overview of the multi-path protocols and discusses their offered improvements. Then, in Section 4, we present results from the testing performed to compare the performance of different protocols in an emulated environment. Finally, Section 5 concludes the paper and summarizes the open research challenges.

2. Alternate Transport Protocols

This section provides a review of some of the crucial issues with the legacy transport protocols, TCP and UDP, and then discusses how and which of the issues the alternate transport protocols, QUIC and DCCP, address.

Issues with TCP -

A. Handshake latency – TCP, being a connection-oriented protocol, performs a 3-way handshake to establish a data connection between the two endpoints. Since TCP itself does not provide any security functions, most applications, such as HTTPS, require the use of cryptographic protocols, such as TLS, that provide privacy, integrity, and authenticity functions. This layering of security functions on top of transport functions leads to increased connection-establishment latencies due to additional handshake round trips. Fig. 1 shows the typical handshaking of a TCP + TLS/1.3 connection establishment for (a) first connection to the server, and (b) subsequent connection to the same server. For a first-time connection, there is 2-RTT of handshake latency before data can be requested. For the subsequent connection to the same server, there is 1-RTT of handshake latency before data can be requested. TLS/1.3 itself reduces the handshake latency as compared to TLS/1.2, which required 2-RTT for the TLS handshake, thus making the TCP + TLS/1.2 handshake require 3-RTT before data can be requested. Since a significant number of connections on the Internet, such as most web transactions, are short transfers, these handshake latencies have an adverse impact on user experience.

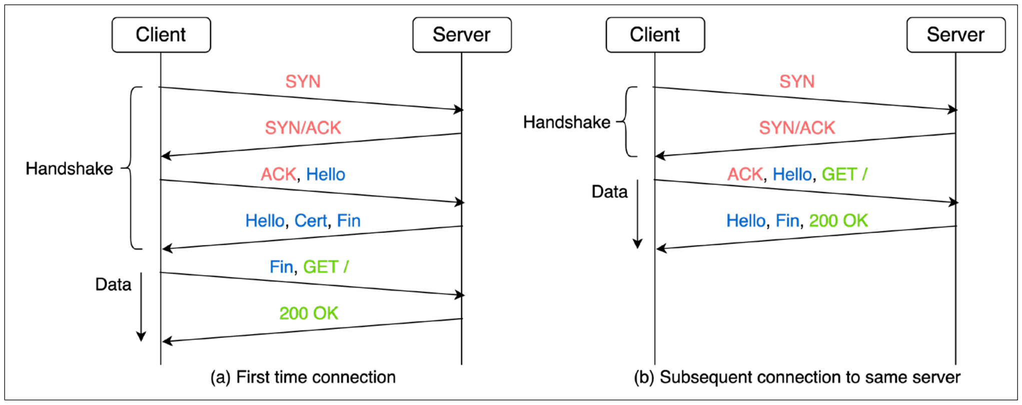


Figure 1 - TCP + TLS/1.3 connection

B. Head-of-line (HoL) blocking – HTTP/2 introduced the notion of multiplexing different HTTP objects via multiple streams onto a single TCP connection. This provided benefits over HTTP/1 by not requiring multiple TCP connections to transfer multiple HTTP objects. However, since for TCP, all application-layer data is just bytestream without having any notion of the application-framing semantics, this results in additional latency incurred for application frames whose delivery needs to wait for retransmissions of previously lost TCP segments. Fig. 2 illustrates this problem – the HTTP endpoints are transferring data using three streams over a single TCP connection. But when TCP packet 2, containing HTTP data of stream 1 is lost, the subsequent TCP packet 3, containing HTTP data of streams 2 and 3, which are independent of stream 1, needs to wait until packet 2 is received, due to TCP's guarantee of in-order delivery, before it can be received by the receiving HTTP endpoint. This negatively impacts the performance of applications running over TCP.

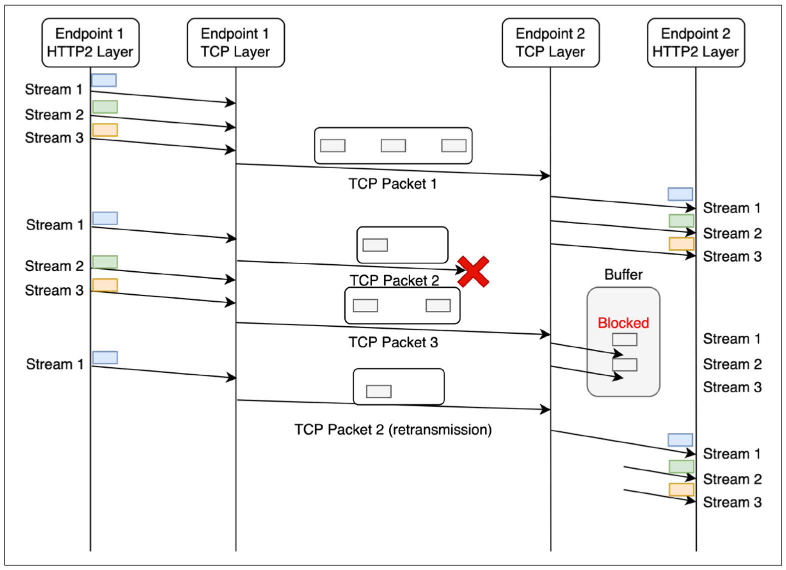


Figure 2 - TCP Head-of-line (HoL) blocking

C. Cleartext transport headers – TCP headers transported between two endpoints are not encrypted. Using TLS, the application data, such as HTTP headers and payload, are encrypted, while the TLS headers and TCP headers remain in cleartext. Fig. 3 depicts a typical HTTP packet with TLS/TCP. Using TLS, the green shaded information is encrypted, the blue shaded TLS headers are visible but tamper-proof, while the unshaded TCP headers are all in cleartext. Research has shown that it is possible to recognize application traffic using the visible transport headers. In [2], a system was developed that could identify the Netflix video being transported over a HTTPS/TCP connection using only the information available in the TCP/IP headers.

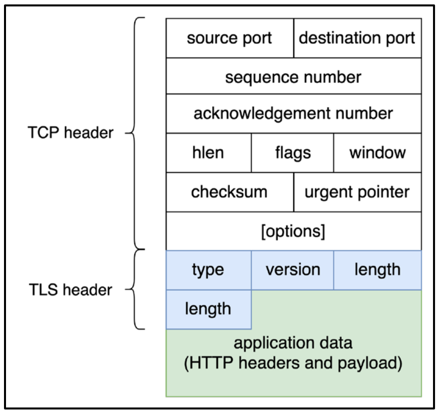


Figure 3 - HTTP with TLS/TCP

UDP is an alternative to TCP for real-time applications that prioritize transport speed over reliability. Since UDP offers a connection-less transport socket, has less overhead as compared to TCP, and is stateless, applications such as VoIP and IPTV prefer it. But UDP offers a very limited set of transport features and lacks certain key attributes – in-order delivery, reliability, flow control, and congestion control. This has led to application developers being restricted to the transport features they can leverage, while having to grapple with the tradeoffs of the two protocols.

Two alternate transport protocols – DCCP and QUIC – have been attempts in the evolution of transport protocols to provide alternatives to either provide a middle ground between TCP and UDP or enhance them to address some of their inherent issues.

2.1. DCCP

DCCP, standardized in IETF RFC 4340, grew out from the observation that while historically UDP was used for short response-request applications, such as DNS and SNMP, the newer applications, such as audio/video streaming and online gaming, were becoming a significant portion of the overall Internet traffic, and having no congestion control transport features posed a problem [3]. Congestion control mechanism control the entry of data packets into the network, enabling better use of a shared network infrastructure and avoiding congestive collapse. While UDP-based applications could implement their own congestion control mechanisms, for example RTCP implementing congestion control for RTP over UDP, a long history of buggy implementations indicates that it is very hard to properly implement effective and reliable congestion control mechanisms. DCCP was designed to provide a modular congestion control framework as part of unreliable transport, i.e., DCCP = UDP + congestion control.

DCCP, like TCP, is a connection-oriented protocol with congestion control, however, like UDP, it does not provide reliability, in-order delivery, or flow control. In other words, DCCP enables loss detection but not loss recovery. Sequence numbers are used in the DCCP packet headers to detect and report losses, but lost packets are not retransmitted. DCCP provides applications with a choice of congestion control mechanisms to choose from, including TCP-like congestion control and TCP-Friendly Rate Control (TFRC). This is negotiated at connection startup via Congestion Control IDs (CCIDs), which refer to one of the standardized congestion control mechanisms.

Applications, such as online gaming, that would prefer making immediate use of available bandwidth and respond quickly to changes in bandwidth, while tolerating abrupt changes in congestion window can use the TCP-like congestion control (CCID 2). While applications, such as media streaming, that would prefer trading off this responsiveness for a steadier, less bursty rate that maintains longer-term fairness with TCP can use the TFRC mechanism (CCID 3).

While DCCP provides an alternate unreliable transport with congestion control, it has seen limited deployments, due to inadequate OS support and issues with NAT-traversal wherein middleboxes sometimes do not understand it. This highlights the complexities associated with designing and deploying alternate transport protocols.

2.2. QUIC

QUIC was initially designed as an experimental transport protocol at Google called GQUIC and was later standardized by the IETF in RFC 9000. The main principles behind QUIC’s design were to improve HTTP performance and subsequently QoE, providing a user space transport that offers more control, and facilitating deployment over existing networks. The designers aimed to develop QUIC as an alternate transport protocol built for the needs of today’s Internet and the modern web, as opposed to a general-purpose transport for most applications. QUIC is built on top of UDP, thereby avoiding the middlebox-traversal issue since most existing networks understand UDP, and it recreates many of the TCP features such as loss recovery, congestion control, flow control, etc. HTTP/3, the latest version of HTTP, runs on QUIC.

QUIC is designed to solve many of TCP’s implicit issues, while also enabling new features, as described below –

A. Low-latency handshake – QUIC, like TCP, is a connection-oriented protocol and needs to perform a handshake before initiating a data session. But QUIC, unlike TCP, has built-in encryption and combines the transport and crypto handshakes to reduce latency. As shown in Fig. 4 (a), it needs 1-RTT of handshake before requesting data, while for a subsequent connection to the same server, it needs 0-RTT of handshake and can request for data in the first packet sent to the server, as shown in Fig. 4 (b). This leads to lower latencies especially for short data transfers that are not impacted by unnecessary handshake RTTs.

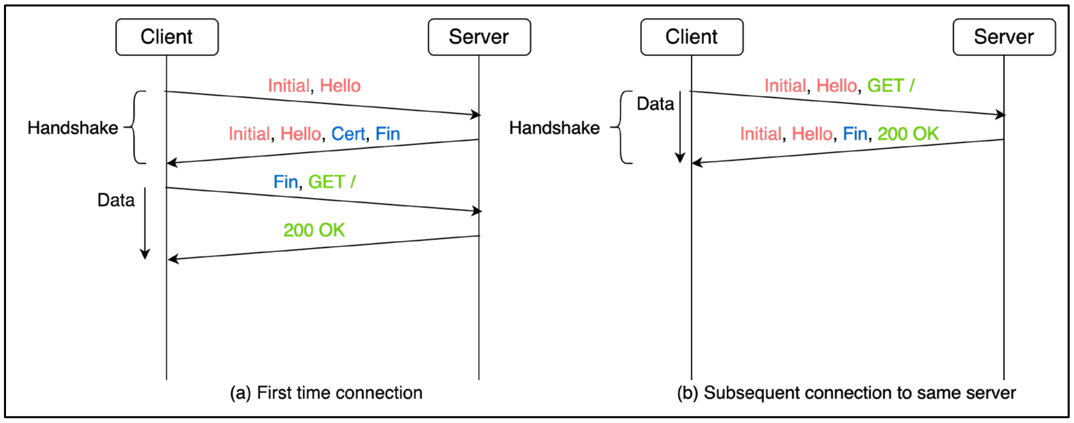

Client
Server
Client
Server
Initial, Hello
Handshake
Initial, Hello, Cert, Fin
Fin, GET /
Data
200 OK
Initial, Hello, GET /
Data
Handshake
Initial, Hello, Fin, 200 OK
(a) First time connection
(b) Subsequent connection to same server

Figure 4 - QUIC connection

B. Stream multiplexing – QUIC uses streams to provide a lightweight byte-stream abstraction to an application. Each stream is identified by a Stream ID and is independent with respect to ordering and retransmissions. Multiple streams are multiplexed inside one QUIC packet. As Fig. 5 illustrates, three HTTP/3 streams are transferring data using QUIC. Since QUIC, unlike TCP, has the notion of streams, lost QUIC packet 2, containing data from stream 1 only, does not block the delivery of the subsequent QUIC packet 3 containing data from streams 2 and 3. This helps applications avoid the HoL problem present in TCP, wherein an affected stream leads to data from all other unrelated streams being affected. This provides improved performance especially in imperfect network conditions where packet losses can severely impact QoE.

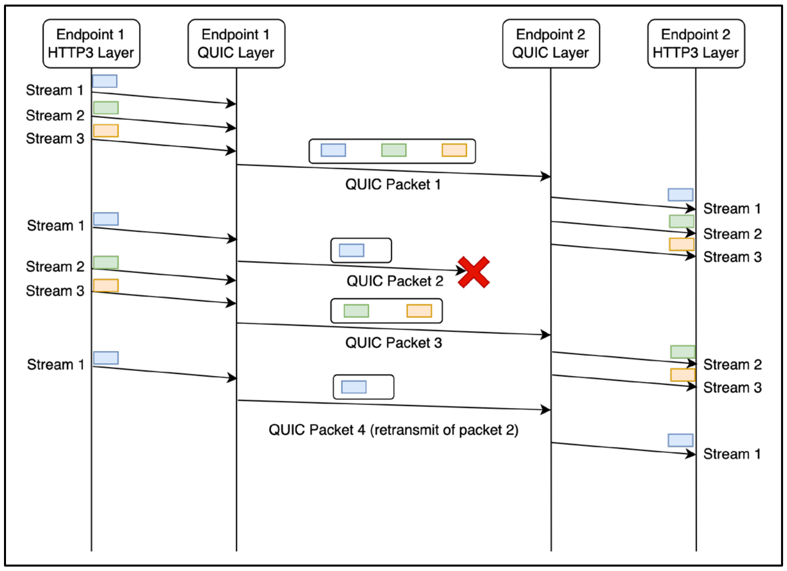


Figure 5 - QUIC's solution to HoL blocking

C. Encrypted headers – QUIC has TLS built in to encrypt all application data and the important headers. Fig. 6 shows a typical QUIC packet wherein all the green shaded information is encrypted, the blue shaded information is visible but tamper-proof, while the unshaded information is in cleartext. Only those QUIC transport headers are visible that are needed for routing or decrypting packets. This encryption of most of the transport headers that were visible in TCP provides two benefits – firstly, it provides privacy by preventing identification of information about the application data by providing end-to-end transport-layer encryption, and secondly, it allows the QUIC features to be deployed without the middleboxes tampering with the transport headers allowing QUIC to run natively over UDP. However, from a network engineer's perspective, there is very little information available to diagnose network performance issues.

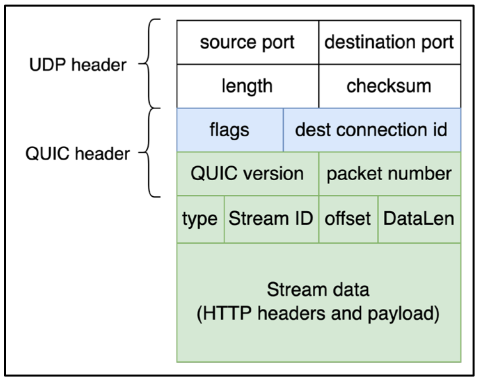


Figure 6 - HTTP with QUIC

D. Connection migration – QUIC provides inherent connection migration by using connection IDs. When a client moves across networks, for example from Wi-Fi to a cellular network, its IP address changes, and the server needs to be notified about the new source IP. In the case of TCP, a new TCP connection must be established from the new source IP, leading to service disruption. QUIC uses connection IDs to identify a connection between the client and server, and this allows changes in lower protocol layers to be handled by routing packets to the same endpoint. A QUIC server provides additional connection IDs to the client during their initial handshake, and the client can use these new connection IDs to maintain service continuity with the server when its underlying network or IP address changes.

These benefits enabled by QUIC are aimed at providing an alternative to TCP with additional built-in features. Operational experiences on QUIC from Google indicated a reduction of 16.7% in Google Search latency at the 99th percentile, 10.6% in YouTube video latency at the 99th percentile, and 18.5% in YouTube video rebuffer rate at the 99th percentile when compared to TCP [4]. However, some caveats exist in these improvements. QUIC's performance benefits over TCP are not consistent across different network types and conditions – benefits are greater in networks with higher average RTT and packet loss, and in desktop clients as compared to mobile clients. Nevertheless, QUIC continues to gain traction with the major websites such as Google, Facebook, Netflix and Snapchat, and the major browsers such as Chrome, Edge, Firefox and Safari all supporting HTTP/3. Currently, about 33% of worldwide secure HTTP traffic is QUIC-based and is expected to grow in the future [12].

3. Multi-path Transport Protocols

Multi-path transport extensions enable a transport connection over multiple network paths simultaneously. For example, a smartphone connected to both Wi-Fi and mobile networks could use the two access networks at the same time, allowing for improved performance and better resiliency. In 3GPP, the ATSSS feature specified for 5G enables this support, wherein a UE can steer a data session over either of the two accesses - 3GPP access or non-3GPP access – enabling best network selection, switch a data session between the two accesses enabling seamless handovers, or split a data session across the two accesses enabling network aggregation, based on the ATSSS rules provisioned by the 5G core [5]. In this architecture, the multi-path client entity is present in the UE, while the multi-path proxy entity is present in the User Plane Function (UPF) in the 5G core. This model enables using the multi-path functionality without having to rely on the application server also supporting the multi-path extensions. The two transport layer-based functionalities specified by 3GPP are Multipath TCP (MPTCP) and Multipath QUIC (MPQUIC). Broadband Forum (BBF) and CableLabs have also specified this ATSSS functionality for a hybrid CPE, also called a 5G-RG, that incorporates both the 5G UE and wireline access functionality [6,7].

3.1. MPTCP

MPTCP is IETF-specified extensions, in RFC 8684, to TCP to enable simultaneous use of multiple network paths between two endpoints. MPTCP was designed to be backward compatible with TCP and with the assumption to either one or both endpoints could be multihomed. MPTCP operates at the transport layer and is transparent to both the upper and lower layers, as shown in Fig, 7, and be able to traverse through middleboxes that understand TCP without requiring any change. MPTCP uses subflows, which are TCP sessions on individual network paths, as part of the larger MPTCP connection.

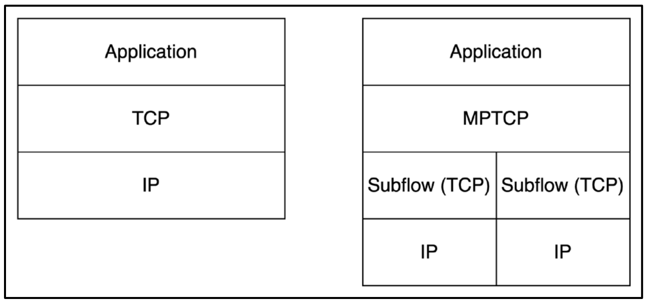


Figure 7 - Comparison of single-path TCP and MPTCP stacks

Fig. 8 shows the order of operations to set up an MPTCP connection. It begins similar to a TCP 3-way handshake, with the difference that the MPTCP-capable hosts add an MP_Capable flag as part of the SYN packets sent to their peers. If both hosts support MPTCP, they further exchange keys and negotiate MPTCP options to enable the establishment of additional subflows. When the client decides to initiate another subflow, for example when the Wi-Fi interface becomes available, another TCP 3-way handshake is initiated from the second interface/port with an MP_Join flag and tokens derived from the keys exchanged during the first handshake. This enables the server to associate the new connection with the first TCP connection, and now the client can use both subflows to transfer data. Each subflow contains sufficient control information, such as the data sequence number, for it to be reassembled and delivered reliably and in-order to the recipient application.

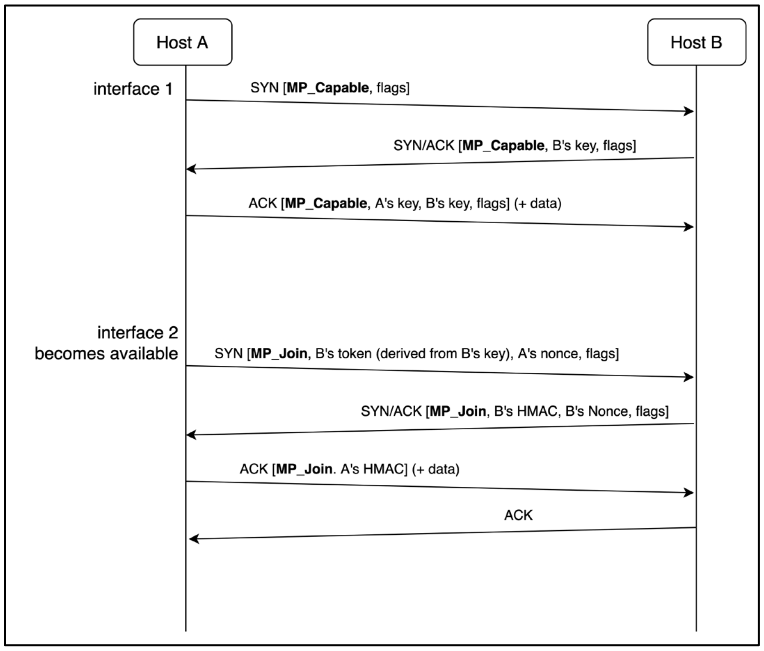

Host A
Host B
interface 1
SYN [MP_Capable, flags]
SYN/ACK [MP_Capable, B's key, flags]
ACK [MP_Capable, A's key, B's key, flags] (+ data)
interface 2 becomes available
SYN [MP_Join, B's token (derived from B's key), A's nonce, flags]
SYN/ACK [MP_Join, B's HMAC, B's Nonce, flags]
ACK [MP_Join. A's HMAC] (+ data)
ACK

Figure 8 - MPTCP operation to set up subflows

MPTCP is supported natively in the Linux kernel and on the iOS/macOS. Apple uses MPTCP for its Siri digital assistant, Apple Maps and Apple Music applications to benefit from its handover capabilities, such as when the user moves away from a Wi-Fi access point and the traffic is handed over to the mobile interface [11].

3.2. MPQUIC

MPQUIC, currently an active IETF draft, is a multipath extension for the QUIC protocol to enable the simultaneous use of multiple paths for a single connection. While the base QUIC protocol supports connection migration between IP-address/port tuples, it can only use one path at a time, while MPQUIC enables the use of multiple paths.

Similar to MPTCP, MPQUIC uses a new transport parameter, initial_max_path_id, to negotiate the use of the multipath extension. To enable additional paths, the endpoints first exchange new connection IDs associated with the other paths, perform path validation to verify reachability of the new IP address/tuple, and then start to use the additional path.

MPQUIC offers several benefits when compared to MPTCP, like what QUIC offers compared to TCP - runs in the user space on top of UDP making its deployment easy with no dependency on OS changes, no HoL blocking caused by lost packets on one stream blocking packets belonging to other streams, and reduced time for subflow establishment. Experiments with video streaming indicate that MPQUIC provides improved QoE as compared to MPTCP with lower rebuffering rates and shorter video startup delay [8]. Testing the delay in handover between Wi-Fi and mobile interfaces, MPQUIC performs similar to MPTCP, with additional benefits of the ability to tune its performance in the user space based on specific requirements [9].

4. Performance Testing Results

In this section we analyze and present the results obtained from testing experiments performed to compare the performance of different transport protocols. The testing was performed in an emulated environment using Mininet running on an Ubuntu server [10]. Throughput testing was performed using iPerf, while the traffic RTT was measured at the client by determining the delay between the sent request and the received response. Each scenario was tested for 100 runs and the results are averaged and presented.

The first test performed was to compare the performance of single-path TCP and MPTCP. Fig. 9 shows the network topology tested and the throughput results. The client is multihomed and is connected via two symmetrical 20 Mbps paths to the server. Throughput tests indicate that while single-path TCP is able to utilize the bandwidth of a single path, MPTCP is able to aggregate the bandwidths of the two paths, resulting in a goodput twice larger than single-path TCP.

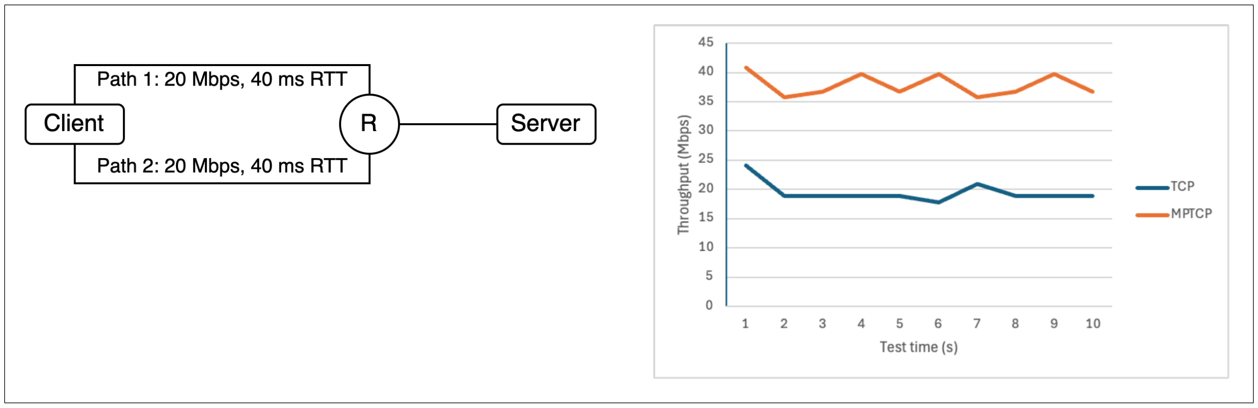
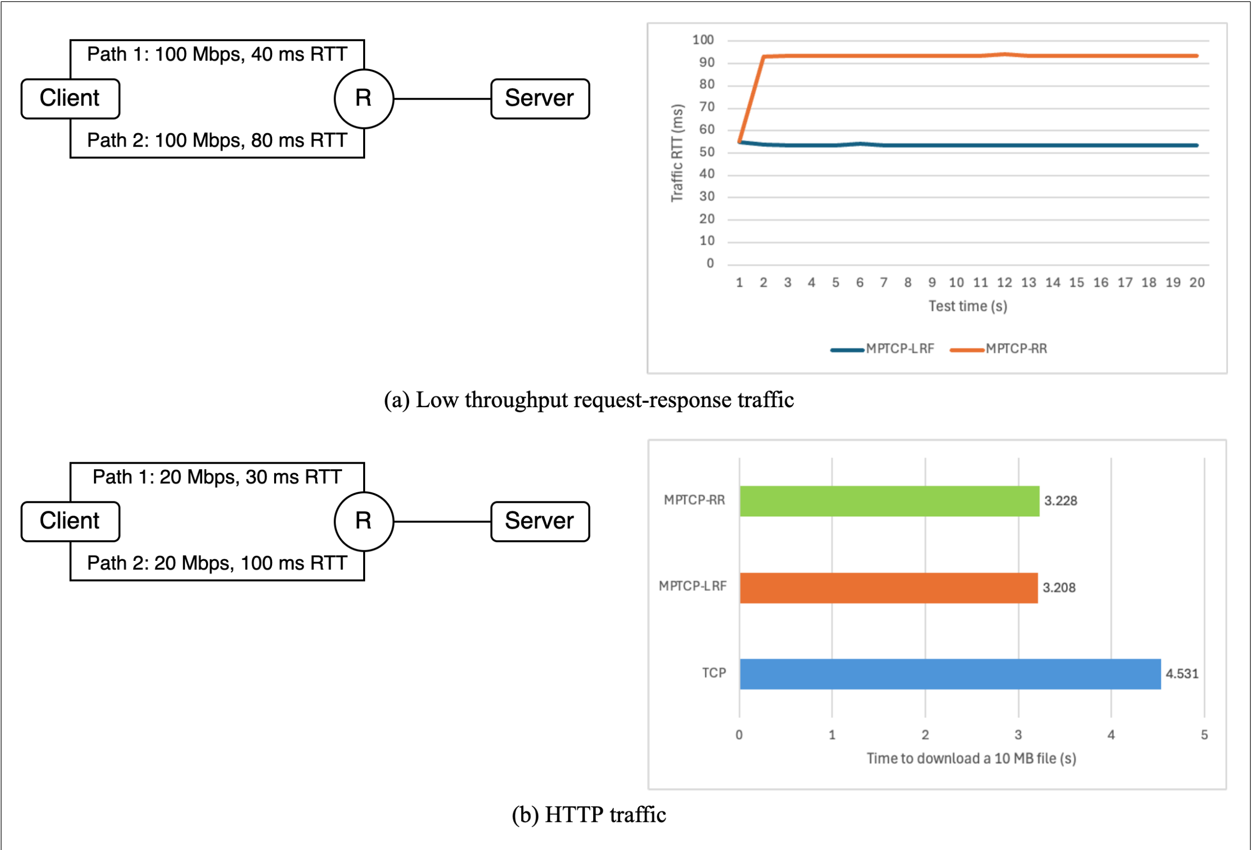


Figure 9 - Single-path TCP vs MPTCP

The next set of experiments performed were to test and compare the three important algorithmic mechanisms of a multi-path protocol – scheduler, path manager, and congestion controller. The packet scheduler is responsible for selecting on which available subflow the next packet will be sent. The two most common packet schedulers are the Lowest-RTT-First (LRF) and Round-Robin (RR). The LRF scheduler, which is the default scheduler in Linux implementations, prioritizes the subflow with the lowest RTT out of all subflows whose congestion window is not yet full. Once the congestion window has been filled, data is then sent on the subflow with the next higher RTT. The RR scheduler selects one subflow after the other, among those which have space in their congestion windows, in a round-robin fashion, striving to evenly utilize the capacity of each path. Some MPTCP implementations use the retransmission and penalization scheme to tackle HoL blocking by retransmitting the lost data segment on an alternate subflow and penalizing the blocked subflow by reducing its sending rate, thereby helping to improve goodput and reduce latency and jitter. Other scheduler implementations include the redundant scheduler which transmits traffic on all available subflows in a redundant way, and the blocking estimation (BLEST) and earliest completion first (ECF) schedulers that aim to increase MPTCP's performance over heterogeneous paths by estimating, once the congestion window of the fastest subflow is full, the tradeoff between sending the next data segment on the slower subflow or waiting for the faster subflow.

To compare the performance of the LRF and RR schedulers, two types of scenarios were tested – low throughput request-response traffic (Fig. 10 (a)) and file download over HTTP (Fig. 10 (b)), over two heterogenous paths. For the first scenario, since the transmitted traffic does not fill the congestion window, the LRF scheduler uses the lower-RTT path to transmit all traffic, while the RR scheduler alternates between the two paths resulting in higher traffic RTT. For the second scenario of bulk transfer, both LRF and RR schedulers outperform TCP, while they perform similarly to each other.



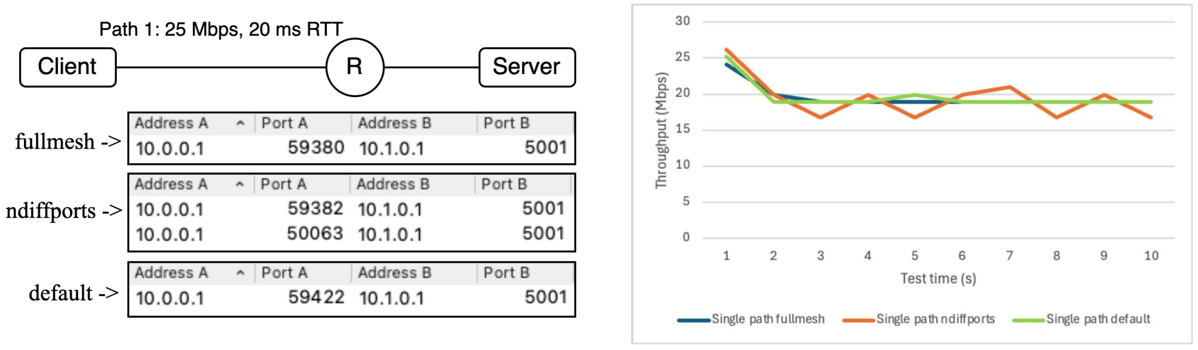
(a) Low throughput request-response traffic

(b) HTTP traffic

Figure 10 - LRF vs RR scheduler

The second aspect to test is the path manager algorithm, which is responsible for determining how subflows will be created over a MPTCP connection. The Linux kernel includes three types of path manager implementations – default, fullmesh, and ndiffports. The default algorithm is a passive path manager that does not initiate any additional subflows on a connection. This is used by the server endpoints that typically would not initiate additional subflows but accept those initiated by the client. The fullmesh algorithm creates a subflow between each pair of (client_IP, server_IP). So, if a client has N addresses and the server M addresses, this path manager will establish N*M subflows. The ndiffports algorithm creates multiple subflows (as per configuration) over the same pair of (client_IP, server_IP) by using different source ports. This path manager was designed considering a specific use case – benefit from multiple equal cost paths in datacenter networks, wherein Equal-cost multi-path (ECMP) routing could be employed to forward packets to a single destination over multiple paths.

Fig. 11 shows three different scenarios tested to compare the performances of the default, fullmesh and ndiffports path managers – (a) single-homed client, single-homed server, (b) dual-homed client, single-homed server, and (c) dual-homed client, dual-homed server. For each scenario, the network topology, Wireshark captures of the TCP conversations between client and server, and the throughput results are presented. The default path manager establishes only one subflow irrespective of the number of available paths, the fullmesh path manager establishes num_client_IP*num_server_IP subflows, while the ndiffports path manager always establishes two subflows (as per configuration) between the same client IP and server IP using different source ports. For the first scenario, the three path managers perform similarly since there is only one path between the client and server. For the second scenario, as expected, the fullmesh path manager, which utilizes both paths, achieves double the goodput as compared to ndiffports and default path managers, which use only a single path. Similarly, for the third scenario, the fullmesh path manager establishes four subflows, one per (client_IP, server_IP) pair, and achieves double the goodput as compared to ndiffports and default path managers, that use only one path. For the fullmesh path manager, similar throughput is observed for scenarios b and c, even though scenario c has more available paths, because of similar bandwidths of the bottleneck links in the two scenarios.



fullmesh ->

Address A	Port A	Address B	Port B
10.0.0.1	59380	10.1.0.1	5001

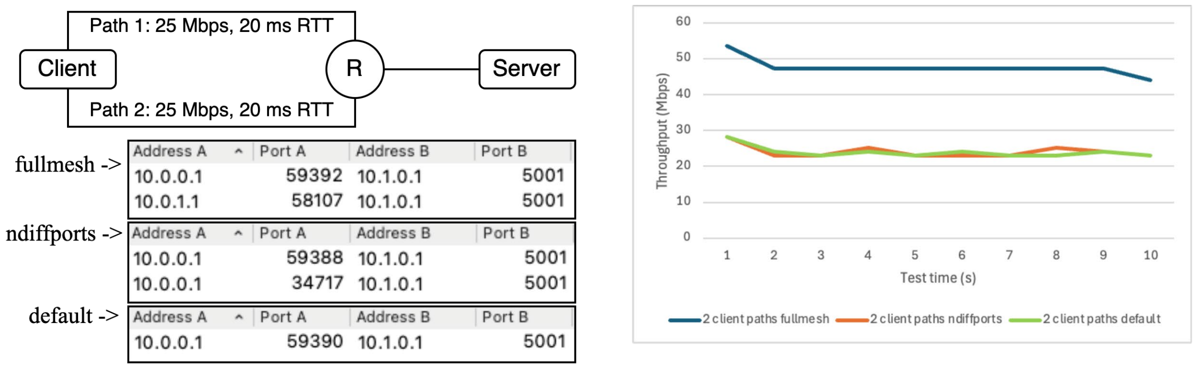
ndiffports ->

Address A	Port A	Address B	Port B
10.0.0.1	59382	10.1.0.1	5001
10.0.0.1	50063	10.1.0.1	5001

default ->

Address A	Port A	Address B	Port B
10.0.0.1	59422	10.1.0.1	5001

(a) Single-homed client, single-homed server



fullmesh ->

Address A	Port A	Address B	Port B
10.0.0.1	59392	10.1.0.1	5001
10.0.1.1	58107	10.1.0.1	5001

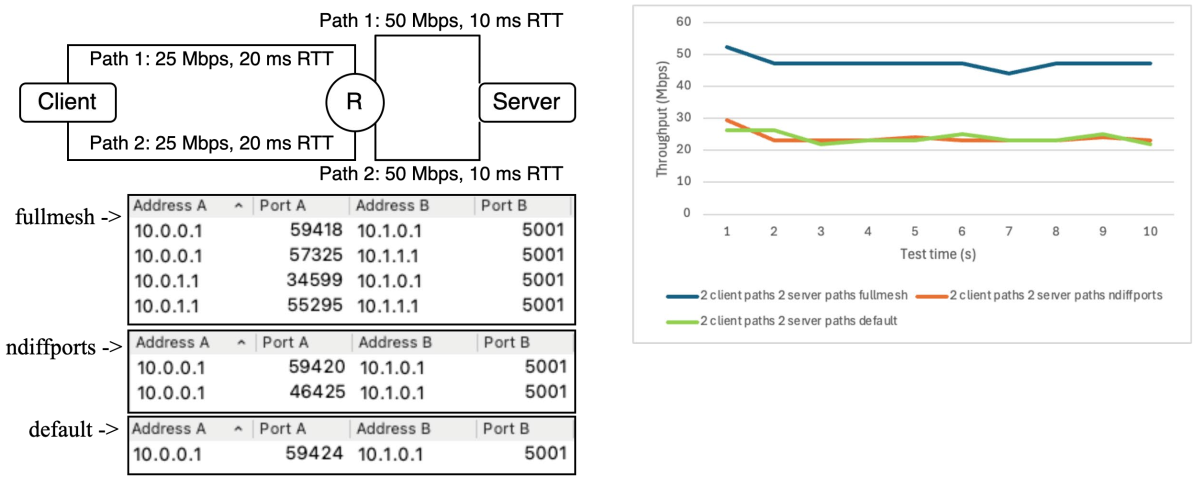
ndiffports ->

Address A	Port A	Address B	Port B
10.0.0.1	59388	10.1.0.1	5001
10.0.0.1	34717	10.1.0.1	5001

default ->

Address A	Port A	Address B	Port B
10.0.0.1	59390	10.1.0.1	5001

(b) Dual-homed client, single-homed server



fullmesh ->

Address A	Port A	Address B	Port B
10.0.0.1	59418	10.1.0.1	5001
10.0.0.1	57325	10.1.1.1	5001
10.0.1.1	34599	10.1.0.1	5001
10.0.1.1	55295	10.1.1.1	5001

ndiffports ->

Address A	Port A	Address B	Port B
10.0.0.1	59420	10.1.0.1	5001
10.0.0.1	46425	10.1.0.1	5001

default ->

Address A	Port A	Address B	Port B
10.0.0.1	59424	10.1.0.1	5001

(c) Dual-homed client, dual-homed server

Figure 11 - Fullmesh vs ndiffports vs default path managers

MPTCP also has the notion of path priorities - regular or backup. This is helpful in case where the two paths do not have the same cost, for example one is an expensive data-limited cellular connectivity versus a flat-cost Wi-Fi connection. In such scenarios, one path can be declared as a backup path to be used only when there are no regular paths available. This path priority is especially important when considering handover scenarios and energy consumption of the end device. In case of two regular paths, subflows are established and data is exchanged on both paths, while in case of one regular and one backup path, subflows are established on both paths while data is exchanged only on the regular path until it fails, detected through successive retransmissions.

Fig. 12 compares these two scenarios, with path 1 having an RTT of 40 ms while path 2 having an RTT of 30 ms, and after 9 seconds of the test, path 1 is triggered to fail by configuring 100% packet loss on that interface. This situation mimics a mobile phone moving out of the coverage of a Wi-Fi access point. In case of two regular paths, the default scheduler LRF uses path 2 having lower RTT for sending data, and a failure in path 1 has no impact. In case of one regular path (path 1) and one backup path (path 2), the regular path is used initially for sending data, even though it has a higher RTT, and when the regular path fails, data is handed over to the backup path, after a delay caused by the time taken to detect path failure.

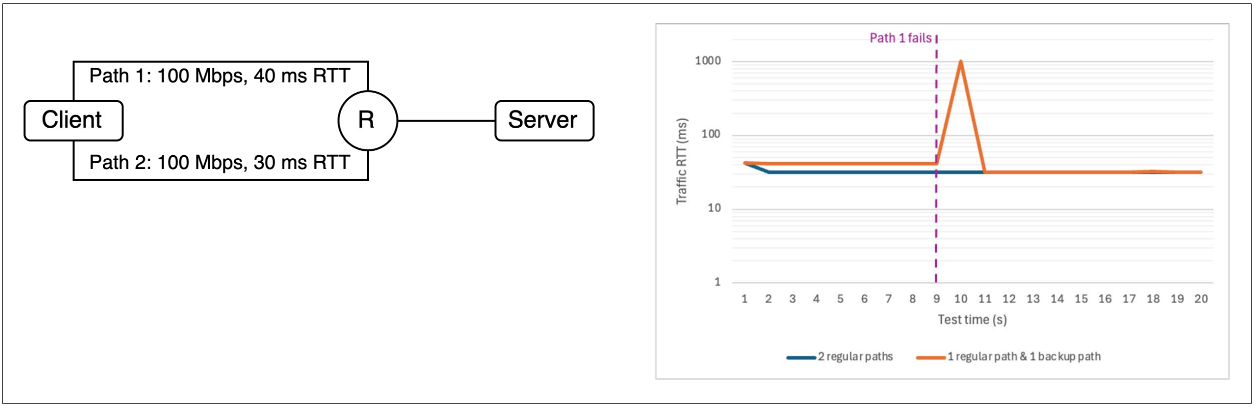


Figure 12 - Handover considering regular and backup paths

The final aspect of multi-path protocols tested was the impact of the congestion control algorithm. The congestion control algorithm is responsible for preventing network congestion and maintaining smooth data flows, while ensuring fairness between different data sessions over a shared link. One of the primary design principles of MPTCP was that its use should not harm other single-path TCP connections over a shared link, i.e., it should not consume more from any of its resources shared by its different paths that if it was a single-path flow. Two types of congestion control algorithms exist – uncoupled and coupled. Uncoupled algorithms such as Reno, CUBIC, Vegas, etc. are designed for single-path TCP, while coupled algorithms such as LIA, OLIA, BaLIA, etc. are designed for multi-path TCP. Since one MPTCP subflow appears as a discrete TCP connection, uncoupled algorithms, that operate independently for each TCP subflow, are not able to provide fairness when sharing the link with other single-path TCP flows.

Fig. 13 compares these two types of congestion control algorithms. The network topology consists of the Client running MPTCP sharing each bottleneck link with another host. The test involved three iPerf flows being generated – first is the MPTCP flow between Client and Server that lasts 50 seconds, second is the TCP Cubic flow between Client-1 and Server-1 that starts 10 seconds after the first flow and lasts 20 seconds, and third is the TCP Cubic flow between Client-2 and Server-2 that starts 20 second after the first flow and lasts 20 seconds. So, the first MPTCP flow competes with the second TCP flow for the upper bottleneck link between 10s and 30s, and competes with the third TCP flow for the lower bottleneck between 20s and 40s. Two congestion control algorithms were tested – Reno and OLIA. The Opportunistic Linked Increases Algorithm (OLIA) is a coupled algorithm that aims to improve throughput as well as a single-path TCP connection on the best available path while having no adverse impact on other single-path TCP connections when sharing common bottlenecks. Two scenarios were tested – (a) the MPTCP connection creating only one subflow on each path, and (b) the MPTCP connection creating four subflows on each path. For the first scenario, Reno and OLIA perform similarly, sharing network bandwidth between the two TCP flows on each path. But for the second scenario, wherein five TCP flows (one single-path TCP flow and four MPTCP subflows) share a path, Reno performs significantly unfairly to the single-path TCP flows. While OLIA, that is designed to ensure that the multi-path goodput, which includes all its subflows, is equal to the single-path goodput, shares the network bandwidth fairly between MPTCP and single-path TCP.

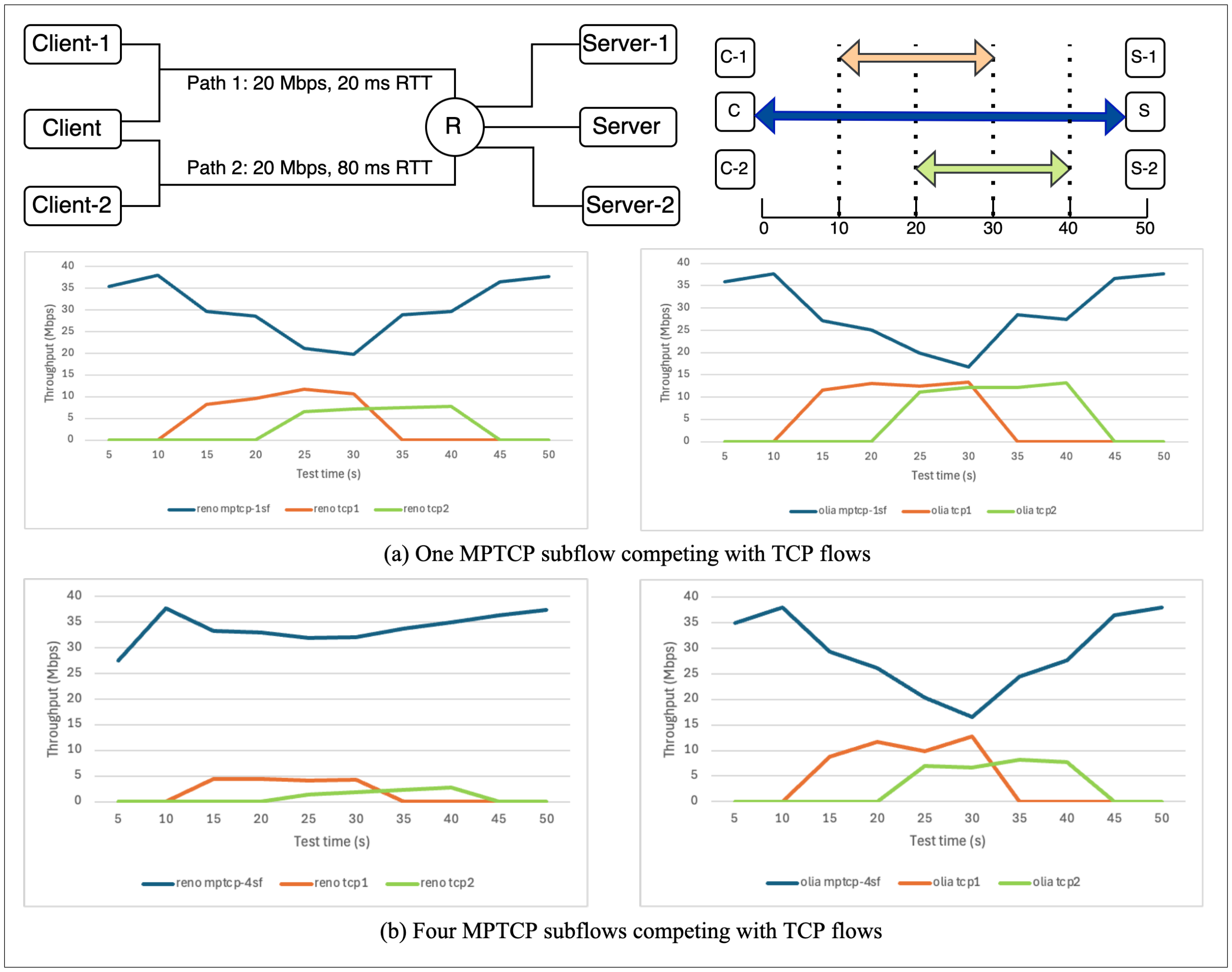


Figure 13 - Uncoupled vs coupled congestion control in multi-path scenario

Like MPTCP, MPQUIC promises improved performance and better network resiliency by leveraging multiple network paths. However, MPQUIC standardization work is still ongoing and there are limited implementations available. Fig. 14 compares the performance of single-path QUIC and MPQUIC when downloading a 5 MB file using HTTP. MPQUIC can aggregate the network bandwidths of the two paths and offers double the goodput as compared to single-path QUIC. Both LRF and RR MPQUIC schedulers outperform QUIC, while they perform similarly to each other for this bulk transfer scenario.

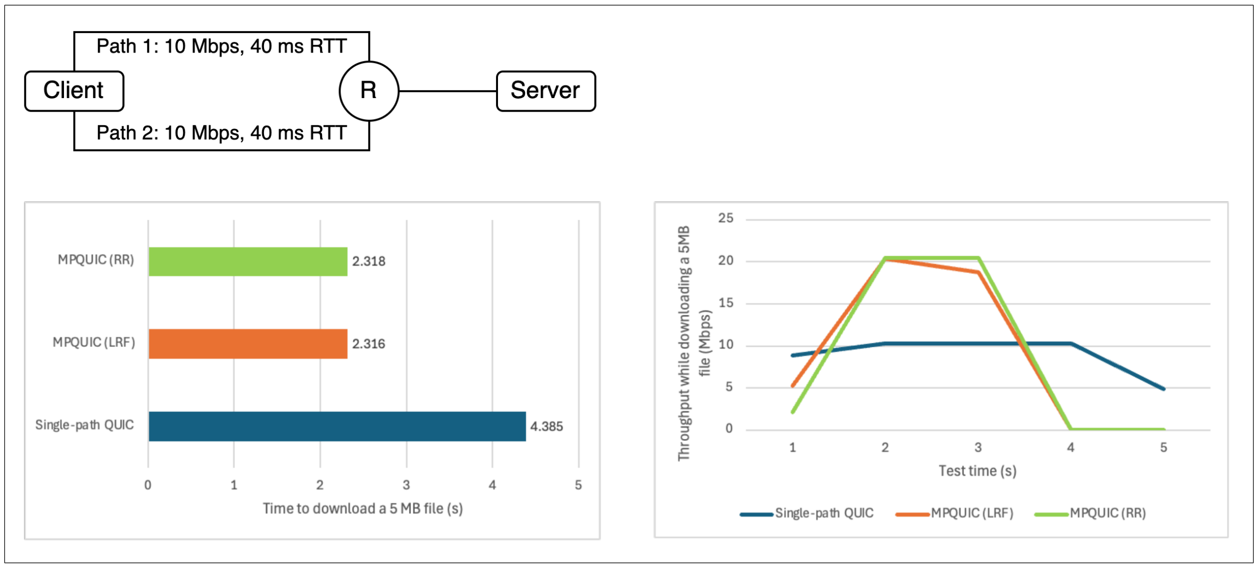


Figure 14 - Single-path QUIC vs MPQUIC

5. Conclusion

With the newer applications demanding increased performance and service continuity across different access networks, this paper first discusses the inherent issues with the currently deployed transport protocols TCP and UDP which makes them unsuitable, and then describes the transport-layer research in developing alternate protocols and multi-path enhancements. While currently the onus is on applications to work around the limitations of TCP and UDP, such as HTTP enabling multiplexing different streams over one TCP connection, or RTP using RTCP to provide congestion control mechanisms over UDP, transport protocols such as QUIC and DCCP provide mechanisms to resolve existing issues and offer additional features. DCCP provides a modular congestion control framework over UDP providing applications the choice to select and use TCP-like or TCP-friendly mechanisms. QUIC offers TCP features like in-order delivery, reliability, flow control and congestion control, while running on top of unreliable UDP. QUIC also fixes some of TCP's issues related to HoL blocking, high handshake-RTT, and cleartext headers, offers additional enhancements such as connection migration, and is gaining widespread traction. The multipath extensions to TCP and QUIC, MPTCP and MPQUIC, respectively, augment their functionalities to utilize multiple available network paths between two endpoints. They allow for improved user experience by enabling best network selection, seamless handover, and network aggregation.

To evaluate the performance of multipath protocols and their different aspects, testing was performed in an emulated environment and the results are presented. MPTCP and MPQUIC provide real benefits when compared to their single-path flavors. Additionally, different kinds of packet schedulers, path managers, and congestion control algorithms were tested to determine their suitability. The experiment results indicate that using the LRF scheduler, the fullmesh path manager, and a coupled congestion control algorithm such as OLIA provides the best performance for most use cases.

The future work for this analysis involves – (*i*) performing testing in real-world network conditions, (*ii*) testing additional aspects such as protocol overhead, computational overhead, and (*iii*) measuring handover delays when switching between Wi-Fi and cellular networks using QUIC, MPTCP and MPQUIC.

Different network operators can make use of these transport-level enhancements in several different ways. Mobile network operators can utilize the ATSSS feature by deploying a multi-path proxy functionality in their core networks that allows their subscribers' multi-path enabled devices to be served over both cellular and fixed accesses. Converged network operators can additionally deploy hybrid CPEs with both cellular and fixed-network capabilities, to provide multi-path capabilities to their subscribers' home devices. Operators can also potentially monetize these enhancements by offering high-performance, seamless handover capabilities as a higher-tier subscription.

In addition to the analysis presented in this paper, CableLabs, working with its members, performed in-house testing for seamless connectivity when transitioning between the Wi-Fi and cellular networks, and researching techniques to resolve the stickiness issue of UE's sticking on Wi-Fi too long before handing over to using a better cellular network for data.

Abbreviations

3GPP	3rd Generation Partnership Project
5G-RG	5G Residential gateway
API	application programming interface
ATSSS	Access Traffic Steering, Switching and Splitting
BBF	Broadband Forum
CPE	customer premises equipment
DCCP	Datagram Congestion Control Protocol
HoL	Head-of-line
HTTP	Hypertext Transfer Protocol
IPTV	Internet Protocol television
LRF	Lowest-RTT-first
MPQUIC	Multipath QUIC
MPTCP	Multipath TCP
OLIA	Opportunistic Linked Increases Algorithm
QoS	quality of service
RR	Round-robin
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
TFRC	TCP-Friendly Rate Control
TLS	Transport Layer Security
UDP	User Datagram Protocol
UE	user equipment
UPF	User Plane Function
VoIP	Voice over Internet Protocol
XR	extended reality

Bibliography & References

[1] G. Papastergiou et al., "De-ossifying the Internet transport layer: A survey and future perspectives," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 619–639, 1st Quart., 2017.

[2] A. Reed and M. Kranch, "Identifying HTTPS-protected Netflix vides in real-time," in *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*, pp. 361-368, Mar. 2017.

[3] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 27-38, 2006.

[4] A. Langley et al., "The QUIC transport protocol: Design and Internet-scale deployment," in *Proceedings of the Conference of the ACM special interest group on data communication*, pp. 183-196, Aug. 2017.

[5] 3GPP, TS 23.501, "System architecture for the 5G System (5GS)," v19.0.0, Jun. 2024.

[6] BBF, TR-470, "5G Wireless Wireline Convergence Architecture," Issue 2, Mar. 2022.

[7] CableLabs, WR-TR-5WWC-ARCH, "5G Wireless Wireline Converged Core Architecture," V03, Jun. 2020.

[8] W. Yang, J. Cao, and F. Wu, “Adaptive video streaming with scalable video coding using Multipath QUIC,” in *Proceedings of the 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pp. 1-7, Oct. 2021.
[9] Q.D. Coninck and O. Bonaventure, “MultipathTester: Comparing MPTCP and MPQUIC in mobile environments,” in *Proceedings of the 2019 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 221-226, Jun. 2019.
[10] Mininet: Rapid prototyping for software-defined networks, https://github.com/mininet/mininet.
[11] O. Bonaventure, “Apple Music on iOS13 uses Multipath TCP through load-balancers,” Oct. 2019, http://blog.multipath-tcp.org/blog/html/2019/10/27/apple_music_on_ios13_uses_multipath_tcp_through_load_balancers.html.
[12] Cloudflare Radar, Adoption & Usage, https://radar.cloudflare.com/adoption-and-usage.