TECC: Towards Efficient QUIC Tunneling via Collaborative Transmission Control

Jiaxing Zhang[1,2], Furong Yang[1*], Ting Liu[1,2*], Qinghua Wu[2,3✉], Wu Zhao[1], Yuanbo Zhang[1]*
Wentao Chen[1], Yanmei Liu[1✉], Hongyu Guo[1], Yunfei Ma[1], Zhenyu Li[2,3]
[1]Alibaba Group, [2]University of Chinese Academy of Sciences, [3]Purple Mountain Laboratories, China

Abstract

In this paper, we present TECC, a system based on collaborative transmission control that mitigates the mismatch of sending behavior between the inner and outer connections to achieve efficient QUIC tunneling. In TECC, a feedback framework is implemented to enable end hosts to collect more precise network information that is sensed on the tunnel server, which assists the inner end-to-end connection to achieve better congestion control and loss recovery.

Extensive experiments in emulated networks and real-world large-scale A/B tests demonstrate the efficiency of TECC. Specifically, compared with the state-of-the-art QUIC tunneling solution, TECC significantly reduces flow completion time. In emulated networks, TECC decreases flow completion time by 30% on average and 53% at the 99th percentile. TECC also gains a reduction in RPC (Remote Procedure Call) request completion time of 3.9% on average and 13.3% at the 99th percentile in large-scale A/B tests.

1 Introduction

Internet privacy has become a more and more serious concern for all Internet residents today. Although policymakers have been making regulations (e.g., GDPR [1]) on Internet privacy stricter than ever to protect users from the abuse of their personal information, these regulations may only be respected by legitimate organizations (e.g., large service providers such as Google), not by malicious hackers on the Internet. Therefore, many knowledgeable people resort to privacy-preserving technologies to further secure their surfing on the Internet.

iCloud Private Relay [2] (PR) is a new privacy protection service announced by Apple at its developer conference (WWDC) in June 2021, which aims to protect customers' Internet activities initiated via the Safari web browser. It uses an architecture with two layers of proxies to guarantee that no one in the middle can directly correlate the user and the target service accessed by the user. In PR's architecture, users' traffic is first routed to an Apple-controlled ingress proxy and the ingress proxy then forwards the received traffic to an egress proxy hosted by third parties (e.g., Cloudflare, Akamai, etc) that finally initiates connections to the target hosts. As Apple has a considerable amount of share on the global smartphone market and PR has a much lower entry barrier compared with traditional privacy-preserving technologies such as Virtual Private Networks (VPNs), Proxies, and The Onion Router (Tor), a significant usage increase of PR is envisioned [3, 4].

To steer user traffic to an egress proxy via an ingress proxy, a tunneling protocol called MASQUE (Multiplexed Application Substrate over QUIC Encryption) is used in PR. In this MASQUE use-case, a client first initiates a QUIC-based HTTP/3 connection to a tunnel server (an ingress proxy) and uses an extended HTTP CONNECT method [5] to open a UDP tunnel towards a target server (an egress proxy). Afterward, the client can communicate with the target server using QUIC (UDP-based) through the opened tunnel. This ultimately leads to a QUIC-in-QUIC communication pattern, as shown in Figure 1.

However, as disclosed by recent studies [4,6], such a QUIC-in-QUIC communication pattern enabled by MASQUE faces several performance challenges. MASQUE only provides a tunneling mechanism and leaves the impact of tunneling on the performance of end-to-end (E2E) connections unexplored. There are also a few recent studies [7–11] that discussed using MASQUE or similar QUIC tunneling schemes to enhance the performance of E2E connections, e.g., accelerating the recovery of packets lost at the last-mile wireless link. Nonetheless, there remains a lack of systematic measurement of MASQUE, which is of great importance for designing efficient QUIC tunneling. To fill the gap, we first conducted a thorough measurement study to dissect the performance of MASQUE in depth. We then identify two fundamental problems that lead to sub-optimal performance:

- *Retransmission in the tunnel:* While retransmission in the tunnel is crucial for accelerated loss recovery, it may cause duplicated retransmissions by both the tunnel server and the server, leading to bandwidth waste and exacerbated network congestion.
- *Congestion control (CC) in the tunnel:* CC in the tunnel is necessary as it can reduce the tunnel's retransmission rate and alleviate network congestion. However, it causes a sending rate mismatch between the inner and outer connection, which can severely degrade E2E throughput.

To tackle these issues, we propose TECC which makes the end hosts and the tunnel server collaboratively control their transmission. In a nutshell, TECC works as follows: network information collected on the tunnel server is fed back to the server through the "Tunnel Server->Client->Server"

* Co-first authors

Corresponding authors: Qinghua Wu (wuqinghua.mail@gmail.com), Yanmei Liu (miaoji.lym@alibaba-inc.com)

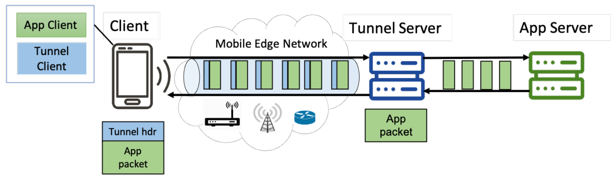


Figure 1: The QUIC-in-QUIC communication pattern enabled by QUIC tunneling

path, which aligns the server's retransmission and sending behavior with the tunnel server. At the same time, TECC offloads the server's CC to the tunnel server, reducing the control loop from the original E2E path to the path between the tunnel client and tunnel server.

We implement TECC and integrate it with the Taobao Android app. Extensive experiments in emulated network environments and real-world large-scale A/B tests demonstrate its efficiency. To sum up, our contributions are:

- We conducted a set of experiments to systematically evaluate MASQUE. An in-depth analysis based on the results reveals the impact of transmission modes (using QUIC streams or datagrams for tunneling E2E packets), retransmission, and CC on the performance of MASQUE.
- Motivated by our key observations on the fundamental limitations of MASQUE, we propose a collaborative transmission control system, TECC, which coordinates the retransmission and CC of the inner and outer connections to achieve enhanced performance.
- We carried out extensive experiments in networks emulated by Mahimahi [12] and large-scale A/B tests in which over 10 million users were involved. Compared with the vanilla MASQUE, TECC decreases the flow completion time (FCT) by 30% on average and by 53% at the 99th percentile in emulated networks. In production networks, TECC reduces the FCT of RPC requests by 3.9% on average and 13.3% at the 99th percentile.

2 Background

2.1 MASQUE overview

MASQUE is a set of specifications developed at the IETF MASQUE working group, which defines new tunneling mechanisms that can leverage QUIC to tunnel E2E UDP and IP packets. Compared with previous tunneling mechanisms (e.g., TCP-based or IP-based tunneling), it benefits from the superior features of QUIC, e.g., 0-RTT connection establishment, more accurate RTT estimation and loss detection, and elimination of head-of-line (HoL) blocking between independent data streams.

Currently, there are two released RFCs in MASQUE: RFC 9297 of "HTTP Datagrams and the Capsule Protocol" [13] and RFC 9298 of "Proxying UDP in HTTP" [5]. The former defines HTTP Datagrams that are the carriers of E2E packets in the tunnel. HTTP Datagrams can be used for conveying multiplexed and potentially unreliable datagrams (e.g., UDP or IP packets) inside an HTTP connection (a tunnel connection). Note, if the tunnel connection is based on HTTP/3 over QUIC, HTTP Datagrams can be transmitted in two different modes: **stream mode** and **datagram mode**. In the stream mode, QUIC streams are utilized to reliably transmit HTTP Datagrams, whereas QUIC datagrams [14] are leveraged to unreliably convey HTTP Datagrams in the datagram mode. The latter specification defines the signaling protocol to set up UDP tunnels inside an HTTP connection. Specifically, depending on the HTTP version of the tunnel connection, a tunnel client can request the tunnel server to open a UDP tunnel via sending an extended HTTP CONNECT request [15,16] or an HTTP Upgrade request [17] with the IP and port of the target server encoded in the request. If the tunnel server accepts such a request, the tunnel is established and the packets of the E2E connection can be transmitted over the opened tunnel. As the tunnel server knows about the addresses of both the client and server, it can forward E2E packets to the correct targets. At the time of writing, there are also a few other specifications under development, e.g., the signaling protocol to establish IP tunnels [18] and the QUIC forwarding mode [19] to reduce the wire and encryption overhead.

In this paper, we mainly focus on the QUIC-in-QUIC usecase enabled by MASQUE UDP tunnels, which means both the tunnel connection and the E2E connection are based on QUIC. We refer to the stream mode and the datagram mode as **MST** (MASQUE stream tunnel) and **MDT** (MASQUE datagram tunnel) in the following paragraphs respectively.

2.2 The potential issues of MASQUE

Although the MASQUE specifications provide functional mechanisms for QUIC-in-QUIC tunneling, the impact of such tunneling mechanisms on the performance of the E2E QUIC connection remains unclear.

First, the two tunnel modes, MST and MDT, could have different impacts on the E2E performance. In MST, each UDP tunnel uses one QUIC stream for the underlying transmission. As the QUIC stream is reliable, if the tunneled E2E packets are lost in the tunnel segment, they can be recovered by the tunnel connection, which is faster than the E2E retransmission. This benefit is not provided by MDT as QUIC does not retransmit QUIC datagrams in which E2E packets are encapsulated. But, MST also has a disadvantage compared with MDT. Due to the QUIC stream only delivering in-order data to its upper layer, MST has a significant HoL blocking problem where a lost packet leads to all subsequent packets being blocked in the tunnel. In MDT, as the QUIC datagrams are independently delivered to the upper layer, the HoL problem does not manifest. Consequently, there are some attempts (e.g., [7]) trying to combine the advantages of MST and MDT by enabling retransmission for MDT (**RMDT**). Nevertheless, a quantitative comparison between these tunnel modes is essential to understand how they perform in practice.

Second, in MST and RMDT, there may be excessive retransmissions as both the tunnel connection and the E2E connection can retransmit packets lost in the tunnel segment. Excessive retransmissions may exacerbate the congestion in the tunnel and waste bandwidth. Applying CC on the tunnel connection may alleviate this problem, but, it also introduces two layers of nested CC, as the E2E connection also performs CC. The nested CC may lead to unwanted sending behavior mismatches between the tunnel connection and the E2E connection.

3 A deep dive into MASQUE

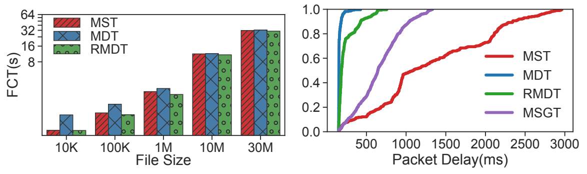
In this section, we first evaluate the performance of different tunnel modes. Then, the necessity of CC in the tunnel is assessed. Finally, as both the tunnel connection and the E2E connection have their own CC, the impact of nested CC is studied. All experiments in this section are conducted in networks emulated by Mahimahi [12].

3.1 The performance of MASQUE

Observation 1: HoL blocking in MST degrades throughput and bloats packet delay. Retransmission in the tunnel is crucial for reducing the complete time of short flows.

To evaluate the performance of different tunnel modes, we transferred files of different sizes through MST, MDT, and RMDT tunnels and calculated the average FCT of the E2E transfers. In the experiments, the E2E RTT and the RTT of the tunnel segment are 200 ms and 100 ms respectively. The loss rate of the tunnel segment is 10%, which emulates a congested link. As the E2E QUIC connection is certainly congestion-controlled (using BBR [20]), the CC of the tunnel connection is disabled in the experiments.

The results are shown in Figure 2a. As observed, MST achieves better performance for short flows compared with MDT, as MST can recover packets lost between the tunnel client and the tunnel server faster than MDT via local retransmission of the tunnel connection. However, as the file size increases, the performance gain of MST is gradually diminished. The diminished return is a result of the HoL blocking in MST as we will explain in the next paragraph. RMDT does the same as MST regarding faster loss recovery, but it performs better than MST as it does not suffer from HoL blocking. We also measured the E2E packet delay of the three tunnel modes, and the results are shown in Figure 2b. Compared with that of MDT and RMDT, the packet delay of MST is higher, which comes from the impact of HoL blocking. Because RMDT also retransmits lost packets, the delay of some packets appears longer from the server's view, as these packets have been retransmitted in the tunnel connection. This explains that the packet delay of RMDT is higher than MDT.



(a) FCTs of E2E transfers (b) CDF of E2E packet delay

Figure 2: The performance of different tunnel modes

Analysis of HoL blocking in MST: When an E2E packet is lost in an MST tunnel, all subsequent E2E packets that have been sent via the tunnel are blocked in the QUIC stream and thus cannot be decapsulated until the lost packet is retransmitted by the tunnel connection. Consequently, the sender of the E2E connection may consider all packets sent after the lost packet as lost, which appears as a huge burst of packet losses. The burst of losses misleads the sender to reckon that it is in a persistent congestion event and it should significantly cut off its sending rate. As there are more packets in large transfers (thus more losses) and large transfers are more sensitive to throughput, the impact of MST's HoL blocking is more evident. Note that the failure of timely decapsulation of packets blocked in the QUIC stream also leads to increased E2E packet delay. Using multiple QUIC streams for the underlying transmission of an MST tunnel can alleviate the impact of HoL blocking, but, our experience indicates that it is still worse than RMDT. Specifically, we extend MST to use a group of QUIC streams (MASQUE stream group tunnel, MSGT). The packet delay of MSGT is between that of RMDT and MST, as shown in Figure 2b.

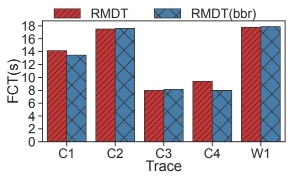
In conclusion, the earlier retransmission in the tunnel provided by MST and RMDT is indeed useful, especially for short flows. As MST suffers from the HoL blocking problem, RMDT seems the more desirable tunnel mode in practice.

3.2 The necessity of CC in the tunnel

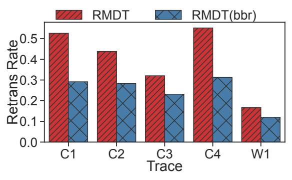
Observation 2: Applying CC on the RMDT tunnel significantly reduces retransmissions and bandwidth waste. But, the E2E throughput does not benefit from the reduction in retransmissions.

In RMDT, packets lost in the tunnel could be retransmitted by both the tunnel connection and the E2E connection. Thus, there is a risk that the tunnel connection sends excessive packets to exacerbate the congestion in the tunnel segment as well as waste bandwidth resources. Ideally, enabling CC on the tunnel connection can reduce such a risk and boost performance.

We conducted experiments to compare the performance of RMDT with BBR and RMDT without CC. In the experiments, the E2E RTT and the RTT of the tunnel are 140 ms and 40 ms.



(a) FCT



(b) Retransmission rate

Figure 3: The comparison between RMDT with and without CC

The loss rate of the tunnel is 10% and the bandwidth bottleneck is put on the tunnel segment. The bottleneck bandwidth is emulated by Mahimahi which replays traces collected from real-world networks. More details about the traces can be found in §5.1. The flow size in the experiments is 30 MB.

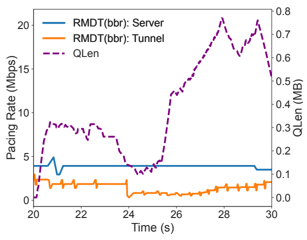
The average FCT and the overall retransmission rate (including retransmissions of the tunnel and E2E connection) are reported in Figure 3b. Indeed, RMDT with BBR significantly reduces retransmissions compared to RMDT without CC. When CC is disabled on the tunnel connection, all lost packets are retransmitted immediately into the network that may already be congested, which in turn triggers more retransmissions and aggravates network congestion. Despite the reduced retransmissions, no significant gain in FCT is observed in Figure 3a. We will see later that this is an outcome of the side effects of nested CC.

3.3 The impact of nested CC

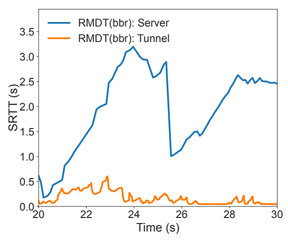
Observation 3: Nested CC causes mismatched sending rates between the tunnel connection and the E2E connection, leading to an increase in the queue length at the tunnel server and bloated E2E smoothed RTTs (SRTTs).

Using the same emulated network environments as in §3.2, we launched long-lived transfers to investigate the impact of nested CC. Note that both the tunnel connection and the E2E connection employed BBR as their CC algorithm. During the transfers, the pacing rate and the SRTT of both the tunnel and E2E connection were recorded. In addition, the sending queue length of the tunnel connection (QLen) was also logged. We present the result of one of the transfers in Figure 4, as the conclusion drawn from the results is generic across different runs.

In Figure 4, we can clearly observe that there is a sending rate mismatch between the tunnel server and the server, leading to persistent queuing at the tunnel server and bloated E2E SRTTs. When packets are sent from the server to the client, the CC of the tunnel server controls the sending rate of the tunnel connection. However, if the sending rate of the tunnel server is lower than that of the server, the tunnel server will inevitably queue packets. The queuing time will increase as the rate difference grows. At the 24th second, the tunnel segment experiences a sudden drop in bandwidth without significant



(a) Pacing rate



(b) SRTT

Figure 4: The impact of nested CC: the pacing rate and SRTT of the tunnel and E2E connection are depicted.

changes in SRTT. The tunnel server can quickly detect this drop in bandwidth and decrease the pacing rate accordingly, whereas the server cannot. This is mainly because the increase in SRTT prolongs the bandwidth updating cycle of BBR in the E2E connection.

3.4 Summary and implication

Our experiment results indicate that RMDT with CC is the more desirable tunnel mode in practice. First, it provides better loss recovery via retransmission in the tunnel while not suffering from the HoL blocking problem of MST. In addition, it limits the risk of sending excessive retransmissions to exacerbate network congestion in the tunnel segment by applying CC on the tunnel connection. However, the two layers of nested CC introduce sending behavior mismatches between the tunnel connection and the E2E connection, which ultimately leads to sub-optimal performance. This calls for coordinating the sending behavior of the two connections to eliminate such mismatches.

In fact, the tunnel server provides opportunities for the coordination of the tunnel and E2E connection. As the tunnel server is closer to the client than the server, it is able to sense the network conditions of the tunnel segment, which is usually the bottleneck (the last-mile access link resides here), in a more accurate and timely manner. The sensed network conditions can be fed back to the server, which assists the server in aligning its sending behaviors with the tunnel server.

4 TECC design and implementation

In this section, we present TECC, a collaborative transmission control system, where the server updates its sending rate based on the feedback from the tunnel server, enabling the efficient coordination of the sending behaviors of the server and the tunnel server.

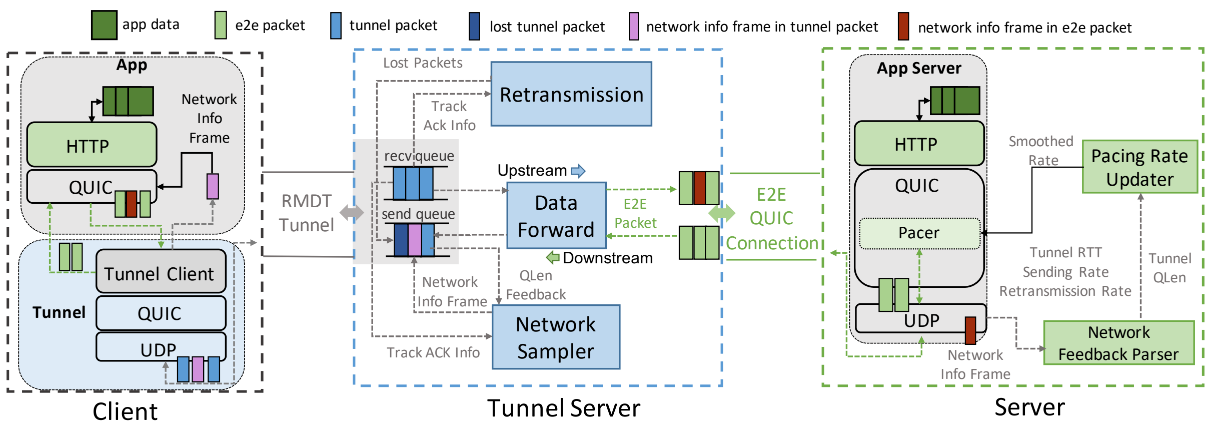


Figure 5: The overview of TECC

4.1 Overview

First, we present an overview of TECC as shown in Figure 5. TECC is based primarily on the RMDT mode and includes three components: the client, the tunnel server, and the server:

Client: Client application modules are mainly responsible for sending and receiving E2E connection data packets. This application uses the QUIC protocol, and the generated QUIC packets will be sent to the tunnel module for encapsulation preparation for entering the tunnel connection. Conversely, QUIC packets received from the tunnel connection will also be sent to the application module. The tunnel module will pass the collected tunnel network info frames to the application module, which will then encapsulate the frame into a QUIC network info frame and pass it to the E2E server application module.

Tunnel Server: The tunnel server mainly employs RMDT. There are three main modules in the tunnel server as shown in Figure 5. The data forward module is responsible for forwarding E2E packets to the client and the server. The retransmission module detects whether the tunnel packets are lost and retransmits those lost packets. The network sampler module is placed in the tunnel server in order to collect important information about tunnel networks and transmission status. The module subsequently generates and transmits network info frames containing the collected information to the client.

Server: The server is mainly responsible for maintaining E2E application modules, and its sending behavior is controlled by the feedback information of the tunnel network. When receiving network feedback information from the client, it parses different network information and uses it for sending rate updates.

4.2 Collaborative transmission control

TECC is based on the RMDT framework. The tunnel server uses the tunnel connection to provide feedback on detected tunnel network information, as well as its own sending and queuing status to the client. After receiving the feedback, the client inserts it into the E2E QUIC frame. Finally, the server updates its sending rate through network feedback of the QUIC frame. TECC achieves two major goals: 1) By quickly collecting bottleneck bandwidth in the tunnel server, the server can respond promptly to dynamic network changes. 2) Through feedback on the retransmission and queue status in the tunnel server, the server can decrease its rate for draining queues, thus reducing queuing delay.

4.2.1 Inaccurate CC in the server

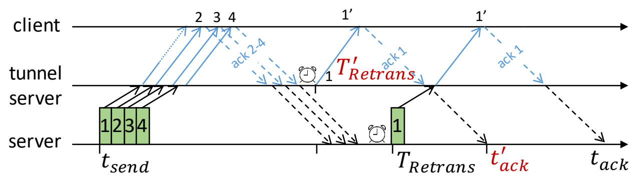


Figure 6: Illustration of retransmission with Tunnel

ACK packets play a vital role in providing network state measurements for server congestion control algorithms. However, due to the impact of RMDT, these packets can inevitably suffer from delays. As shown in Figure 6, assuming the RTT from the client to the server is T_s, and the tunnel is T_t. The retransmission detection time in the server and tunnel is $T_{Retrans}$ and $T'_{Retrans}$, respectively. At the time $t_{send} = 0$, the server transmitted packets with sequence numbers 1-4, but the packet with sequence number 1 was lost in the tunnel link. If the tunnel server does not retransmit, the server receives the ACK packet with sequence number 1 at $t_{ack} = T_{Retrans} + T_s$. However, if the tunnel server retransmits the packet, the server receives the retransmitted ACK packet at $t'_{ack} = 1/2(T_s - T_t) + T'_{Retrans} + T_t + 1/2(T_s - T_t) = T'_{Retrans} + T_s$. However, since the server does not have the knowledge of whether the tunnel server has retransmitted the lost packets, the delays calculated for the packet(s) at the time t'_{ack} is $Delay(t'_{ack}) = t'_{ack} - t_{send} = T'_{Retrans} + T_s$. And for packets retransmitted by the server, the delay is $Delay(t_{ack}) = t_{ack} - T_{Retrans} = T_s$. In addition, the delay in the tunnel server also increases extra response time. Therefore, these can result in an incorrect evaluation of delay by the server, which affects the accuracy of the server-side congestion control.

4.2.2 Tunnel feedback

To solve the problem of inaccurate CC, we propose a collaborative feedback control framework that offloads the server-side CC to the tunnel server. This framework proactively perceives network information in the tunnel link close to the client and provides feedback to the server for sending rate estimation.

There are two main factors that affect the server's network assessment in the Tunnel Server: retransmission and congestion control. Retransmission causes the lost packets in the tunnel server to be delayed at least T_t after the client responds with an ACK packet, and the signal of packet loss is transformed into a signal of delayed packet delay. Therefore, we need to collect the proportion of retransmitted packets in Tunnel Server $r(t)$ at time t and the min RTT T_t of the tunnel server. Meanwhile, the delay in ACK perception will lead to a slower response of the server to the changes in the tunnel network. Therefore, transmitting the tunnel server's sending rate $Tr(t)$ can more quickly perceive the probe of the tunnel network. The congestion control of the tunnel server makes the queuing situation more severe. Therefore, recording the length of the queue within the connection in the Tunnel Server $q(t)$ can explicitly reduce the server's rate and empty the queue.

Feedback cycle: To determine the optimal tunnel feedback cycle τ, we may need to conduct experiments and evaluate the trade-offs between the accuracy of feedback and the overhead of sending feedback messages. The RTT between the client and the tunnel server is T_t. In our extensive experiments, setting τ to T_t yields high accuracy at a low cost, therefore we use T_t as the cycle τ for tunnel feedback.

4.2.3 Server's pacing rate updating rule

The server in the cooperative scheme employs a pacing-based CC algorithm to regulate its sending rate. It makes sense to use the pacing mechanism in the tunneling scenario to prevent the accumulation of tunnel server queues caused by burst traffic. Therefore, we use the server's sending rate $Sr(t)$ at time t as an indicator to control the sending of packets. In addition to the sending rate, window-based congestion control can also be supported, with the send control window, $W(t) = Sr(t) \cdot T_s$, limiting the total number of packets sent in an RTT. In order to better understand, the relevant parameters are presented in Table 1.

The congestion control mechanism on the server is triggered when it receives a feedback packet containing tunnel information frame from the tunnel server. After parsing the tunnel information frame, server obtains the latest sending rate $Tr(t)$ of the tunnel server. To ensure that the tunnel server and the server send packets at the same rate, we assign the sending rate $Sr(t)$ of the server to $Tr(t)$.

Parameter	Explanation
$Sr(t)$	server's sending rate
$Tr(t)$	tunnel server's sending rate
$U(t)$	proportional parameter, $U(t) \in [0,1]$
r_{ai}	additive increase part
$q(t)$	queue length of the tunnel server
θ	the time needed for the server to empty its queue
δ	proportional parameter, approximately 1
$r(t)$	retransmission rate of the tunnel server
max_pf	maximum penalty factor on the queue

Table 1: Explanation of parameters

If $Sr(t) > Tr(t)$ and the bottleneck link is the link between the client and the tunnel server, the queue at the tunnel server will be built up rapidly and grow gradually. However, if $Sr(t) < Tr(t)$, the tunnel server is limited by the number of packets sent by the server and may not have enough packets to utilize the available bandwidth. Therefore, to avoid queue creation while maximizing the utilization of bottleneck bandwidth, it is important to ensure that $Sr(t) = Tr(t)$.

Additionally, in traditional congestion control algorithms, the sending rate rapidly increases at the beginning of connection establishment to occupy the bandwidth. However, by congestion control in tunnel, servers pacing as $Tr(t)$ can avoid the data limitation at the early stage of connection establishment to probe the bandwidth at the tunnel server. The server pacing rate is:

$$Sr(t) = U(t) \cdot Tr(t) + r_{ai} \tag{1}$$

Due to bandwidth changes or response delays, the tunnel will inevitably build up a queue. To ensure a consistent rate, it will be impossible to empty the queue completely. Therefore, we use $U(t) \in [0,1]$, which changes in real-time according to the tunnel link, as shown in Equation 1. Additionally, we use the multiplicative-increase/multiplicative-decrease (MIMD) strategy to enable the server to quickly respond to tunnel link changes, and add an additive increase (AI) part to ensure fairness, which decouples link control and fairness. This approach is inspired by HPCC [21].

4.2.4 Penalty for building up queue

In a tunneling scenario, a tunnel server hosts multiple E2E connections simultaneously, which makes it more prone to queuing than traditional E2E connections. This is due to a reduced bandwidth of the tunnel link. Therefore, a penalty mechanism for queue establishment is needed in the tunneling scenario to reduce the sending rate and empty the queue in time. This ensures that the queue at the tunnel server remains almost empty.

There are many traditional congestion control algorithms that guarantee low queue length. For example, DCTCP uses ECN markers to feedback on the queueing of the switch and reduces the sending window by ECN feedback [22]. However, these algorithms have limited transmission information and their performance is limited by ECN thresholds, which makes them inflexible. Additionally, HPCC algorithms can transmit detailed intermediate link load information, which can detect the network queueing situation promptly and accurately for more precise regulation [21]. However, they have complex software and hardware implementations, which makes them difficult to deploy. In the tunneling scenario, the feedback framework can easily feed back the network status and queuing situation to the server for congestion control.

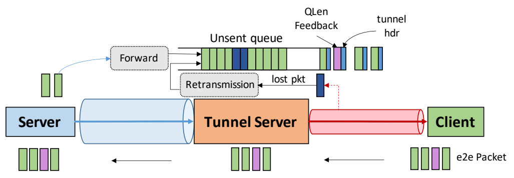


Figure 7: Queueing in the tunnel server

Estimate server pacing rate based on QLen: The tunnel server is responsible for transmitting E2E packets from the server to the client in the downlink, which can result in the buildup of the queue when the server's transmission rate $Sr(t)$ surpasses the rate of transmission $Tr(t)$. As shown in Figure 7, the minimum observed queue length (QLen) within a feedback cycle τ is denoted as $q(t)$, representing the QLen at time t.

The server calculates the pacing rate to drain the queue as:

$$Sr(t) = Tr(t) - \frac{q(t)}{\theta} \tag{2}$$

The parameter θ represents the time needed for the server to empty its queue and is useful in preventing abrupt changes to the sender rate. Usually, θ is set as a fraction of T_s. However, setting a small θ value can lead to inadequate bandwidth for packet probing in the tunnel server, while a large θ value may cause a delay in clearing the queue. To find an appropriate value of θ, we conducted lots of experiments with various values of θ. Eventually, we discovered that it achieves better performance when θ is around $2/3T_s$. Therefore, our research set $\theta = 2/3T_s$.

Alleviate congestion caused by retransmissions: Although retransmission from the tunnel server might decrease the time required for loss recovery, it can increase congestion on the tunnel link. This occurs because retransmitted packets are inserted into the queue of the tunnel server. Therefore, when determining the appropriate queue length, it is important to consider the retransmission of lost packets.

$$Sr(t) = Tr(t) - \frac{q(t) + \delta r(t)Tr(t)}{\theta} \tag{3}$$

The retransmission rate of the tunnel server is represented by $r(t)$. We assume that $r(t)$ remains constant throughout the feedback delay $T_s + d_q$, resulting in $r(t + T_s + d_q) = \delta r(t)$, where δ is approximately 1 (e.g., $\delta = 0.95$). At time $t + T_s + d_q$, the queue length is the sum of retransmitted packets $\delta r(t) \cdot Tr(t)$ and the queue length $q(t)$ at time t.

From Equation 1 and Equation 3, we get

$$U(t) = 1 - \frac{q(t) + \delta r(t)Tr(t)}{\theta Tr(t)} \tag{4}$$

When there is a sudden drop in the bandwidth of the tunnel link, the queue will experience significant growth. Therefore $U(t)$ may decrease to near 0. In order to prevent a rapid decrease in the server rate, We place constraints on the variable $U(t)$ to remain within a specific range:

$$U(t) = \max\{1 - \frac{q(t) + \delta r(t)Tr(t)}{\theta Tr(t)}, 1 - max\_pf\} \tag{5}$$

The symbol max_pf represents the maximum penalty factor on the queue to prevent the server rate from dropping too fast. In later experiments, we set max_pf to 1/2.

Queue length noise filtering: To mitigate real-time queue length fluctuations caused by measurement errors, network delays, and other transient factors, we implement a queue length noise filtering procedure. Such fluctuations may cause the server to overreact and lead to substantial changes in transmission rates, subsequently impacting client application performance. We employ two primary methods. Firstly, we perform multiple detections at the tunnel server to obtain an accurate measurement of the queue length, allowing us to filter out invalid information. Secondly, we use Exponentially Weighted Moving Average (EWMA) to filter the queue ratio $U(t)$ before updating the server's transmission rate. This strategy ensures a smoother, more gradual adjustment of the server's send rate while minimizing the impact on client applications.

4.2.5 Retransmission trigger on the server

There are primarily two ways to trigger packet retransmission on the server: timeout retransmission and fast retransmission. The fast retransmission includes time-based fast retransmission and packet-based fast retransmission. The first type is triggered based on a time threshold, which we set as 9/8 SRTT. The second type is triggered based on receiving subsequent packet ACKs, with an initial threshold value of 3. Due to the retransmissions of the tunnel server, the SRTT of the server should increase. Thus, to suppress duplicated retransmissions, we increase the time-based fast retransmission threshold of the server by T_t. Regarding packet-based fast retransmission, since retransmission by the tunnel server exacerbates the out-of-order client ACK packets, the threshold for reordering increases gradually as spurious retransmission by the server is triggered, leading to a decrease in the probability of server retransmission. Therefore, the server's rate of duplicated retransmission is relatively small and does not exceed 1% in our experiments.

4.2.6 Fairness

In tunnel scenarios, multiple flow connections go through the same tunnel server to different servers on the same bottleneck link. Unlike traditional E2E congestion control algorithms, the bottleneck link detection for tunnel connections relies on the sending rate of the tunnel server, which is dependent on the information of the tunnel connections. However, because the pacing rate update rule uses MIMD strategy, fairness cannot be provided among multiple competing tunnel flows. To achieve fairness among multiple flows, we add an additive-increase (AI) term to the update of pacing rate, making it a multiplicative-and-additive-increase/multiplicative-decrease (MAIMD) scheme. This approach theoretically provides better fairness among multiple competing flows [23–26]. The additive-increase (AI) is set as follows:

$$r_{ai} = \frac{MSS}{T_s} \tag{6}$$

$$Sr(t) = U(t) \cdot Tr(t) + r_{ai} = U(t) \cdot Tr(t) + \frac{MSS}{T_s} \tag{7}$$

The server increases the pacing rate every time it updates by one Maximum Segment Size (MSS) in one server round-trip time, which is a small increase designed to control rate convergence while preventing network congestion. Achieving equilibrium requires that multiple competing tunnel flows pass through the same bottleneck link, resulting in the bottleneck bandwidth being constant between flows. Consequently, the server's sending rate is then modified based on the bottleneck bandwidth such that the sending rates of multiple flows can achieve equilibrium. By reaching equilibrium, fairness is maintained among competing flows.

The server's sending rate depends on the detected bandwidth of the tunnel server. Therefore, TECC's RTT fairness depends on the tunnel server's original congestion control algorithm (wherein we have used BBR [20]). Consequently, the collaboration approach and BBR have varying throughput rates under different tunnel RTT flows [27]. In tunnel scenarios, the original congestion control algorithm's unfairness can be mitigated even further. In §5.4, we demonstrate this through experiments including multiple tunnel flows with various RTTs.

Algorithm 1 Sender algorithm

```
Data: Tunnel Server Feedback: $Tr(t)$, $q(t)$, $T_t$
Result: Server: $Sr(t)$
1 function UpdateSenderRate():
2     $e(t) \leftarrow \frac{q(t)+\delta r(t)Tr(t)}{\theta Tr(t)}$
3     $U(t) \leftarrow \max\{1-e(t), 1-max\_pf\}$
4     $U \leftarrow (1-ewma\_weight) \cdot U + ewma\_weight \cdot U(t)$
5     $Sr(t) = U \cdot Tr(t) + \frac{MSS}{T_s}$
6     return $Sr(t)$
```

The sender's sending rate must consider not only the tunnel server's sending rate but also the current queue length of the tunnel server and fairness. Ultimately, the complete logic of the sender's sending rate is illustrated in Algorithm 1.

4.3 Implementation

This section mainly discusses the methods we deployed TECC as well as the optimizations we made for large-scale deployment.

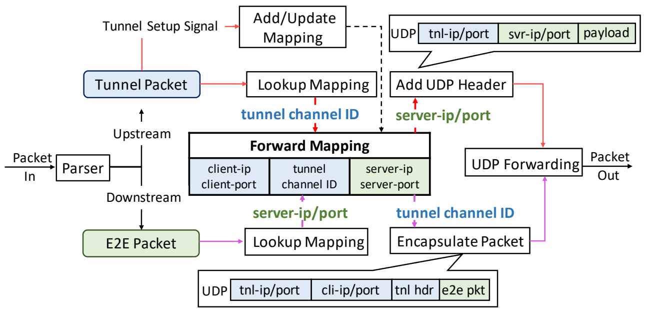


Figure 8: Data forwarding logic in the tunnel server: forwarding downstream/upstream E2E packets

Forwarding in the tunnel server: The packet forwarding mechanism in the tunnel server both encapsulates and decapsulates the E2E packets and directs them to their respective destination addresses, as shown in Figure 8. When a client establishes a tunnel connection to the tunnel server, the server's IP and port are forwarded to the tunnel server as well. After receiving the tunnel setup signal, the tunnel server generates a mapping of the tunnel channel ID that associates the client's IP and port and the server's IP and port. As a result, the E2E packets from the corresponding tunnel channel ID are decapsulated and transmitted to the UDP socket that is bound to the same server IP and port. Correspondingly, the E2E packets from the equivalent socket are encapsulated into the corresponding datagram. Furthermore, the tunnel server facilitates updating forwarded information. If clients require rebinding the forwarding mapping, they can retransmit the tunnel setup signal on the relevant tunnel server.

Retransmission: Retransmission plays a crucial role in enhancing client performance in tunnels. In the downstream link, retransmission can significantly decrease packet loss recovery time in the tunnel connection. Furthermore, the tunnel server can preemptively detect packet loss from the client in the upstream link, enabling packet loss recovery ahead of time on the client side. We follow the retransmission mechanism in RMDT to eliminate blocking and ensure improved performance. Datagram retransmission requires the tunnel to track the lower-layer QUIC datagram's delivery status closely. Selective retransmission of data packets is adopted with indications of packet loss. The experiment in this paper mainly focused on download scenarios that necessitate retransmitting all data packets in the tunnel connection. Nevertheless, some time-sensitive video scenarios permit partial packet loss. Therefore, leveraging the client's application information allows us to retransmit crucial or time-constrained frames selectively, thereby enhancing bandwidth utilization.

Feedback implementation in the tunnel: The feedback in TECC relies on the support of both the QUIC protocol stack and the tunnel protocol stack. To accomplish this, we have developed the TUNNEL_INFO frame and QUIC_INFO frame in the tunnel protocol and the QUIC protocol. These frames are used to transmit feedback information from the tunnel server to the tunnel client and from the app client to the app server. Because both the tunnel client and app client are deployed on our APP, the tunnel client can directly transfer feedback information to the app client when receiving feedback information.

Tunnel server selection: It's important to select the appropriate edge tunnel server since users can experience better performance by selecting the edge tunnel server closest to them. In our app, the client obtains real-time information about the tunnel server from our own DNS. The tunnel client needs to obtain tunnel information, including the protocol version and the tunnel server information from our DNS first. It then establishes a tunnel connection to complete the transmission of the E2E connection through the tunnel packets. Our DNS owns the information of all tunnel servers and selects the tunnel servers that are close to the user and have sufficient resources.

Load balancing: Load balancing (LB) in the tunnel server is essential because the tunnel server has to handle a large number of tunnel requests and forward massive E2E connection packets. In practical deployments, a bidirectional LB strategy is required, which includes packets from the tunnel client and the server. To achieve this, we use multiple workers with NGINX. Due to the address migration in QUIC, the QUIC CID is considered instead of traditional network five-tuples [28]. Since "tunnel connection CID" and "E2E connection CID" don't follow the same generation rules, it can be challenging to balance the packets to the same worker. Therefore, during actual deployment, the CIDs in tunnel QUIC connections and E2E QUIC connections follow the same consistency encoding rules. This allows the same client's tunnel connection and E2E QUIC connection packets to be routed to the same worker.

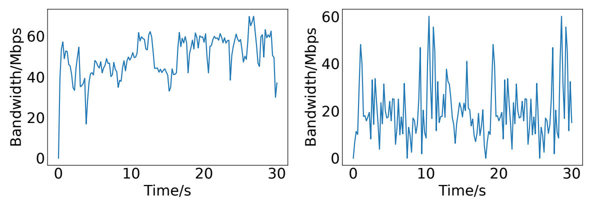
5 Evaluation

In this section, we present the evaluation of the TECC, which consists of two parts: online evaluation and simulated environment evaluation.

Online evaluation: In this part, we collected anonymous data from mobile users who upgraded our app with "QUIC Tunnel". To validate the improvement of TECC compared to the MST, MDT, and RMDT, we conducted a series of large-scale A/B experiments. The user scale of the experiment exceeds ten million. The experimental results show that the TECC significantly reduces completion time.

Evaluation in emulated networks: The Mahimahi-based emulation environment was established to evaluate the performance of the QUIC Tunnel. The test environment comprises three principal containers: a client running HTTP/3 and the tunnel stack, a tunnel server running the tunnel stack, and a server running HTTP/3. Mahimahi emulates the links from the client to the tunnel and from the tunnel to the server. We presume that the link from the client to the tunnel is the bottleneck link, so all test cases mainly involve client requests for a specific file. This download scenario is widely used in several applications, such as web browsing and video downloading.

5.1 Experiment setup



(a) Cellular Home (b) Wi-Fi Home

Figure 9: Bandwidth variation of different traces

Baseline: In the non-tunnel solution, the tunnel server acts as an intermediate forwarding node based on iptables, forwarding packets between the client and the server. MDT and RMDT come from vanilla MASQUE and are detailed in Sec 2. We also compared the forwarding mode (MFT) suggested by Apple to proxy QUIC in MASQUE similar to non-tunnel methods [29]. Since the performance of the non-tunnel solution is similar to MFT, we will refer to both as MFT later. In addition, the RTT between the tunnel server and the server is set to 100ms.

Traces: We used five real-world mobile network traces, four from cellular and one from Wi-Fi, namely: cellu-

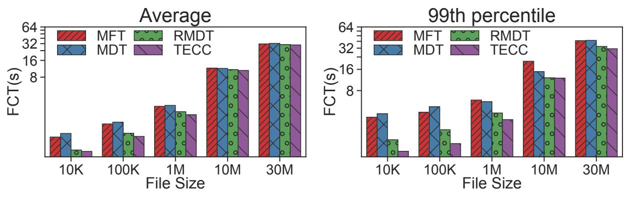


Figure 10: FCT of different solutions under various flow sizes

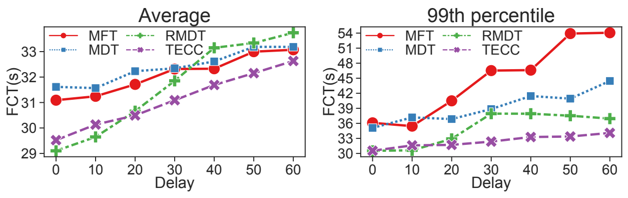


Figure 11: FCT of different solutions under various delays

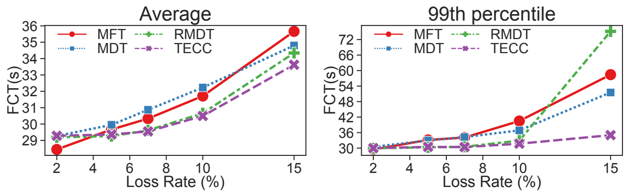


Figure 12: FCT of different solutions under various loss rates

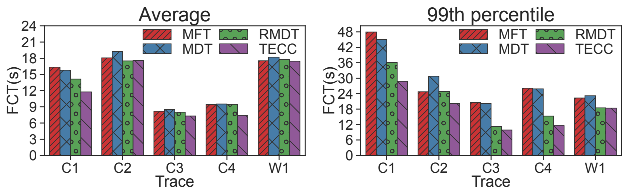


Figure 13: FCT of different solutions in mobile networks

lar_driving (C1), cellular_outdoor (C2), cellular_home (C3), cellular_subway (C4), wifi_office (W1). Among them, the detailed bandwidth information for C3 and W1 is shown in Figure 9. We also measured the performance at stable bandwidth, the bandwidth was set to 10Mbps.

5.2 End-to-end performance

Flow size: *Tunnel is an effective way to reduce the flow completion time, particularly for short flows.* We run the request for different size files from the server on 10Mbps bandwidth, the result shown in Figure 10. The tunnel solutions, including RMDT, and TECC, outperform non-tunneling MFT methods in the case of short flows (flow length less than 10M). The performance of MDT is similar to that of MFT. For short flows, packet loss delays can seriously reduce the overall flow completion time, while the tunnel's early retransmission mechanism can effectively reduce the packet loss recovery time and improve overall performance. As the flow length increases, the delay of a small number of lost packets has less impact on the overall flow completion time, so the benefit of the tunneling mechanism is relatively reduced. In TECC, the end-tunnel cooperative scheme outperforms RMDT. RMDT solves the HoL blocking problem by reducing the latency of blocking but leads to low bandwidth utilization due to a mismatch of the sending rate and bandwidth.

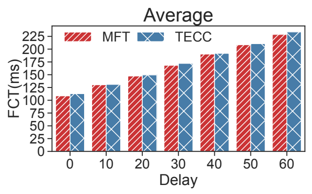
Client-Tunnel delay: *The benefits of the tunnel vary with the change of Client-Tunnel delay and lower delay results in higher benefits.* In our experiment, we transmitted thousands of 30M files with various delays on a 10Mbps network, and the results are presented in Figure 11. As the delay increases, the benefits provided by the tunnel gradually decrease. Compared to MDT, RMDT performs better and still has over 2% optimizations even at 50ms. However, due to rate mismatch, its performance declines when compared to MFT at higher delays. TECC is significantly optimized when compared to the others, and it still maintains good optimization effects even at 60ms. These findings indicate that the TECC continues to deliver certain optimization effects even in mobile edge networks with high RTT.

Loss rate in Client-Tunnel link: *The benefits provided by the tunnel increase as the loss rate increases, especially in tail optimization.* We conducted experiments involving the transmission of thousands of 30M files with varying loss rates on a 10Mbps network, and the results are depicted in Figure 12. When the packet loss rate is low, the additional overhead of the tunnel results in lower performance compared to MFT. However, as the packet loss rate increases to over 5%, the FCT of MDT and MFT are similar. Simultaneously, RMDT and TECC display superior performance compared to MFT. At the 99th percentile, when the packet loss rate exceeds 10%, TECC can reduce FCT by over 21.7%. These findings suggest that in mobile edge networks with a high loss rate, TECC can provide significant throughput enhancements.

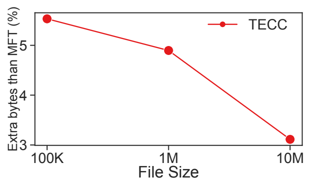
Mobile networks: *TECC exhibits strong optimization effects in mobile networks, achieving an average reduction of over 15% at the 99th percentile.* To evaluate the performance of the tunnel in mobile networks with dynamic bandwidth fluctuations, we conducted tests on a large number of 30MB files under various mobile network traces. A comprehensive introduction to the trace can be found in the appendix. The result is shown in Figure 13. In terms of both average FCT and 99th percentile FCT, TECC outperforms MFT and other tunnel solutions across different network traces. In some cases, the average FCT of TECC was reduced by over 30%, and the 99th percentile saw a reduction of more than 53%. Although RMDT has limitations, they generally perform better than MFT in mobile network environments. This evidence suggests that TECC can perform well in different mobile network scenarios.

5.3 Tunnel overhead

Compared to not using the tunnel, the tunnel incurs additional overhead, primarily involving the setup delay in connection establishment and the extra packet overhead of packet encap-

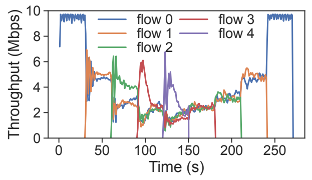


(a) Setup Delay

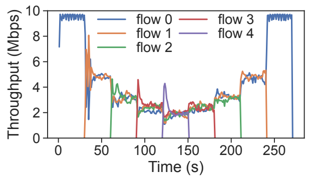


(b) Extra Tunnel Header

Figure 14: Tunnel overhead in networks without random losses



(a) TECC w/o AI



(b) TECC with AI

Figure 15: Fairness among competing tunnel flows

sulation. Consequently, we respectively tested the overhead of the tunnel under different client-tunnel delays and different file request sizes, the result shown in Figure 14.

Setup delay: *The setup delay of the tunnel is higher, but overall it does not exceed 2%.* Owing to the processing overhead of the tunnel protocol stack and the delay in establishing the tunnel connection, the setup delay is slightly elevated. As the delay fluctuates, the increase in setup delay time remains relatively constant, suggesting that the setup delay does not exhibit significant changes with increasing delay.

Extra packet overhead: *The tunnel incurs additional packet overhead, which overall does not exceed 4% in long flow.* Due to the tunnel packet encapsulation in the QUIC datagram, TECC adds a datagram header, QUIC header, and other QUIC frame overhead compared to UDP. With the increase of data flow, the relative proportion of data packet overhead gradually decreases, from only 5% in the 100K to 3% when it reaches 10M.

5.4 Fairness

	MFT (BBR)	TECC (BBR)
Different Server Delay	0.8039	0.9916
Different Client Delay	0.6759	0.8620

Table 2: Jain's Fairness Index among different RTT flows

Flows with the same RTT: First, we demonstrate that the TECC can converge to fair bandwidth allocation for flows with the same RTT upon arrival and departure. We first start a client-to-server flow through a tunnel server, with a fixed

	mean	p95	p99	p999
MST	2.7%	1.9%	6.0%	17.5%
TECC	3.9%	4.5%	13.3%	36.0%

Table 3: Improvements of MST and TECC

10Mbps bandwidth in the tunnel link. Then, we incrementally add a flow from a different client accessing a different server through the same tunnel server. Finally, we start tearing down flows one by one. As shown in Figure 15, without additive increase, the MIMD strategy may exhibit throughput jitter upon flow arrival, as flows in the startup phase of BBR [20] initially preempt the bandwidth, leading to a rapid increase in throughput for newly arrived flows. By adding an additive increase, the competition flow can converge more smoothly, making it less likely for existing flows in the link to be preempted by new flows, resulting in better fairness. By calculating Jain's fairness index, the value is greater than 0.99, ensuring fairness for TECC when tunnel flows with the same RTT enter the same tunnel link.

RTT unfairness: In tunnel scenarios, including the two delays of client-to-tunnel server and tunnel server-to-server links, we conducted fairness experiments on flows with different delays in each link. For different server delays, we launched 8 different flows with server delays of 10-80 ms on the same 10Mbps bandwidth tunnel link, with a fixed client-to-tunnel delay of 10ms. Based on the average throughput of the 8 flows after convergence, we calculated the corresponding Jain's fairness index values. We conducted the same experiment for different client delay, with 8 different client delays of 10-80, and the results are shown in Table 2. We compared the cooperative solution's CC with MFT. Each experiment was run for 10 rounds, and the average index value was taken in the table. As the results show, compared to MFT, the CC of the cooperative solution can achieve RTT fairness on different server delay, because the tunnel detection by the server is mainly completed through the tunnel node, and the bandwidth allocation and detection frequency are related to the tunnel's RTT, thus achieving fairness in flows with different server delays. Different client delays are related to the original tunnel detection congestion control algorithm (BBR [20, 27]), but because the original end-to-end control loop is shortened to the client-to-tunnel, the impact of different RTTs on throughput is reduced, thus mitigating the RTT unfairness problem of the original congestion control algorithm.

5.5 Real-world A/B tests

We compared the completion times for RPC requests from over 10 million users who use vanilla MASQUE and TECC. To investigate the reduction of FCT with TECC, we deployed three different types of tunnels (MST, MDT, and TECC) to users in the same region. At the same time, the tunnel server is deployed at the edge nodes closer to the selected users. In order to measure the effectiveness of TECC, we used the completion time of user requests with MDT as the baseline. The recorded relative improvements of the MST and TECC in reducing completion time are presented in Table 3.

MDT does not optimize the mobile edge network and is equivalent to direct forwarding similar to MFT. Our results demonstrate that, regardless of average or tail completion times, both MST and TECC outperform MDT. Furthermore, due to the HoL blocking problem and mismatch in CC actions in MST, TECC achieved a 7.3% improvement in the 99th percentile and an 18.5% improvement in the 999th percentile compared to MST.

6 Discussion

Multiple E2E connections in tunnel: In real-world network, it is common for a bottleneck link to transmit multiple network connections simultaneously. These different connections compete for bandwidth through their respective bandwidth probing techniques. While using a tunnel to transport multiple E2E connections, in addition to priority rules, the TECC tunnel can dynamically control the bandwidth probing of each E2E connection based on the type of each connection. This can achieve better traffic fairness on mobile edge networks.

Congestion control in tunnel: The congestion control at the tunnel server can prevent traffic congestion by restricting the sending rate of packets. However, this may result in longer delay of packets that have arrived at the tunnel server. In our work, we only compared the results when using CC or not. It is beneficial to design a CC framework which also considers the sojourn time of packets at the tunnel server, to balance the tradeoff between efficient congestion control and short delay. This is left as our future work.

Potential enhancements for MASQUE: Presently, the research efforts in the MASQUE working group primarily concentrate on providing generic solutions to proxy different protocols. However, employing these techniques maybe degrade user performance. TECC is dedicated to addressing the performance issues of MASQUE's generic solutions, but achieving an effective universal standard necessitates the support of MASQUE.

7 Related work

Last-mile performance optimization: For most applications where remote servers are involved, optimizing the wireless network close to the client is crucial for last-mile performance. CDNs cache user-requested resources in edge servers closer to user networks; however, the efficiency of CDNs decreases for processing dynamic data [30–32]. An alternative method is Performance Enhancing Proxy (PEP), which optimizes different feature links by breaking the E2E connection [33]. PEP mostly relies on TCP protocols to achieve performance gains in mobile and satellite networks [34–36]; however, it is not applicable to the header encryption and verification mechanism of QUIC packets. Sidecar [37] proposed an ACK-based protocol that enables the proxy to perceive E2E encrypted data, but servers need to explicitly perceive and communicate with the intermediate proxy, making deployment more difficult and increasing communication overhead. TECC leverages QUIC tunnel's own detection and feedback mechanisms to mitigate PEP's damage to E2E connection semantics and ensure data privacy by sending feedback information to the sender in the direction of the receiver.

Middlebox feedback for rate control: Feedback-based middleware has gained widespread application in optimizing server congestion control algorithms [21, 26, 38–40]. Zhuge [38] shortened the control loop by increasing the delay of ACK packets in wireless Access Points. ABC [26] controls server window size by explicitly marking "accelerate" or "brake" signals based on router's perception of network conditions. HPCC [21] obtains accurate load information and rate control via in-network telemetry (INT). However, these congestion control protocols are mostly based on router or switch information package condition detection of a network. In a tunnel, the middle server can actively detect network conditions and control packet transmission rate, thus providing servers with more rate information.

8 Conclusion

In this paper, we present TECC to optimize the E2E performance of the QUIC tunnel. Deployed at the tunnel server which is closer to the client, TECC provides faster and more precise network feedback to assist servers with a more accurate estimation of the available bandwidth. In emulated networks, TECC decreases flow completion time by 30% on average and 53% at the 99th percentile. TECC also gains a reduction in RPC (Remote Procedure Call) request completion time of 3.9% on average and 13.3% at the 99th percentile in large-scale A/B tests.

Acknowledgements

We thank our shepherd Vaibhav Bajpai and the anonymous reviewers for their constructive feedback. This work was supported in part by the National Key R&D Program of China (2022YFB2901800), Natural Science Foundation of China (U20A20180, 62072437), and Beijing Natural Science Foundation (JQ20024).

References

[1] General Data Protection Regulation (GDPR) – Official Legal Text. https://gdpr-info.eu/.

[2] Apple Inc. iCloud Private Relay Overview. https://www.apple.com/kr/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf, 2021.

[3] Patrick Sattler, Juliane Aulbach, Johannes Zirngibl, and Georg Carle. Towards a tectonic traffic shift? investigating Apple's new relay network. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, pages 449–457, New York, NY, USA, October 2022. Association for Computing Machinery.

[4] Martino Trevisan, Idilio Drago, Paul Schmitt, and Francesco Bronzino. Measuring the Performance of iCloud Private Relay. In Anna Brunstrom, Marcel Flores, and Marco Fiore, editors, *Passive and Active Measurement*, Lecture Notes in Computer Science, pages 3–17, Cham, 2023. Springer Nature Switzerland.

[5] David Schinazi. Proxying UDP in HTTP. RFC 9298, August 2022.

[6] Mirja Kühlewind, Matias Carlander-Reuterfelt, Marcus Ihlar, and Magnus Westerlund. Evaluation of quic-based masque proxying. In *Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC*, pages 29–34, 2021.

[7] Furong Yang, Yanmei Liu, and Yunfei Ma. A Configurable Retransmission Extension for HTTP/3 Datagrams. Internet-Draft draft-yang-masque-dgram-retrans-01, Internet Engineering Task Force, March 2023. Work in Progress.

[8] Zsolt Krämer, Mirja Kühlewind, Marcus Ihlar, and Attila Mihály. Cooperative performance enhancement using quic tunneling in 5g cellular networks. In *Proceedings of the Applied Networking Research Workshop*, pages 49–51, 2021.

[9] Michele Luglio, Mattia Quadrini, Cesare Roseti, Francesco Zampognaro, and Simon Pietro Romano. A quic-based proxy architecture for an efficient hybrid backhaul transport. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 144–146. IEEE, 2020.

[10] A Abdelsalam, Michele Luglio, Mattia Quadrini, Cesare Roseti, and Francesco Zampognaro. Quic-proxy based architecture for satellite communication to enhance a 5g scenario. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2019.

[11] Mike Kosek, Hendrik L. Cech, Vaibhav Bajpai, and Jörg Ott. Exploring proxying quic and http/3 for satellite communication. *2022 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2022.

[12] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[13] David Schinazi and Lucas Pardue. HTTP Datagrams and the Capsule Protocol. RFC 9297, August 2022.

[14] Tommy Pauly, Eric Kinnear, and David Schinazi. An Unreliable Datagram Extension to QUIC. RFC 9221, March 2022.

[15] Patrick McManus. Bootstrapping WebSockets with HTTP/2. RFC 8441, September 2018.

[16] Ryan Hamilton. Bootstrapping WebSockets with HTTP/3. RFC 9220, June 2022.

[17] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.

[18] Tommy Pauly, David Schinazi, Alex Chernyakhovsky, Mirja Kühlewind, and Magnus Westerlund. Proxying IP in HTTP. Internet-Draft draft-ietf-masque-connect-ip-13, Internet Engineering Task Force, April 2023. Work in Progress.

[19] Tommy Pauly, Eric Rosenberg, and David Schinazi. QUIC-Aware Proxying Using HTTP. Internet-Draft draft-ietf-masque-quic-proxy-00, Internet Engineering Task Force, August 2023. Work in Progress.

[20] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[21] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[22] Mohammad Alizadeh, Albert Greenberg, Dave Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Dctcp: Efficient packet transport for the commoditized data center. 2010.

[23] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, jun 1989.

[24] Aditya Akella, Srinivasan Seshan, Scott Shenker, and Ion Stoica. Exploring congestion control. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.

[25] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.

[26] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. Abc: A simple explicit congestion controller for wireless networks. 2020.

[27] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th international conference on network protocols (ICNP)*, pages 1–10. IEEE, 2017.

[28] Martin Duke, Nick Banks, and Christian Huitema. QUIC-LB: Generating Routable QUIC Connection IDs. Internet-Draft draft-ietf-quic-load-balancers-16, Internet Engineering Task Force, April 2023. Work in Progress.

[29] Tommy Pauly, Eric Rosenberg, and David Schinazi. QUIC-Aware Proxying Using HTTP. Internet-Draft draft-pauly-masque-quic-proxy-06, Internet Engineering Task Force, March 2023. Work in Progress.

[30] Volker Stocker, Georgios Smaragdakis, William Lehr, and Steven Bauer. The growing complexity of content delivery networks: Challenges and implications for the internet ecosystem. *Telecommunications Policy*, 41(10):1003–1016, 2017.

[31] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[32] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulié, Christophe Diot, and Pablo Rodriguez. Greening the internet with nano data centers. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48, 2009.

[33] Jim Griner, John Border, Markku Kojo, Zach D. Shelby, and Gabriel Montenegro. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, June 2001.

[34] Carlo Caini, Rosario Firrincieli, and Daniele Lacamera. Pepsal: a performance enhancing proxy for tcp satellite connections. *IEEE Aerospace and Electronic Systems Magazine*, 22(8):7–16, 2007.

[35] Ye Li, Liang Chen, Li Su, Kanglian Zhao, Jue Wang, Yongjie Yang, and Ning Ge. Pepesc: A tcp performance enhancing proxy for non-terrestrial networks. *IEEE Transactions on Mobile Computing*, 2023.

[36] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating transparent web proxies in cellular networks. In *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*, pages 262–276. Springer, 2015.

[37] Gina Yuan, David K Zhang, Matthew Sotoudeh, Michael Welzl, and Keith Winstein. Sidecar: in-network performance enhancements in the age of paranoid transport protocols. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 221–227, 2022.

[38] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 193–206, 2022.

[39] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, 2002.

[40] C-H Tai, Jiang Zhu, and Nandita Dukkipati. Making large scale deployment of rcp practical for real networks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 2180–2188. IEEE, 2008.