PROGRAMA DE DOCTORADO EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES EN REDES MÓVILES

TESIS DOCTORAL

REDUCIENDO LATENCIAS EN PROTOCOLOS DE TRANSPORTE: MEJORAS EN EL PROTOCOLO QUIC

PhD THESIS

REDUCING TRANSPORT PROTOCOL LATENCY: QUIC PROTOCOL ENHANCEMENTS

AUTOR

MIHAIL ZVEREV

DIRECTORES

RAMÓN AGÜERO CALVO

JOSU BILBAO UGALDE

UNIVERSIDAD DE CANTABRIA

Escuela de Doctorado de la Universidad de Cantabria

Santander 2023

Declaration

Ramón Agüero Calvo, PhD, Professor at the University of Cantabria in the Area of Telematics Engineering.

Josu Bilbao Ugalde, PhD, Director of Electronics, Information and Communication Technologies at IKERLAN Technology Research Center.

HEREBY STATE THAT

The thesis entitled "Reducing Transport Protocol Latency: QUIC Protocol Enhancements" has been carried out by **Mr. Mihail Zverev** in the IoT & Digital Platforms team of IKERLAN Technology Research Center, under our supervision, and it meets the requirements for doctoral work.

Santander, May of 2023

Ramón Agüero Calvo

Arrasate/Mondragón, May of 2023

Josu Bilbao Ugalde

Affiliation
IoT & Digital Platforms
Information and Communication Technologies
IKERLAN Technology Research Center

This work was supported by the Industrial Doctorates Program from the University of Cantabria (Call 2018).

Acknowledgements

I would like to thank my supervisors Ramón Agüero Calvo and Josu Bilbao Ugalde for their unwavering support throughout the whole thesis. Despite their busyness, they could always find time to discuss my problems and rescue me from the swamp of my perfectionism. Their guidance was critical in progressing towards the goals. And special thanks for the paternal care during my trips.

Pablo Garrido Ortiz and Fátima Fernández Pérez have been helping me along the way with bright ideas and continuous assistance. Their careful review and constructive criticism helped me get out of the dead ends I was getting into. Without their help I would probably not finish this thesis.

I would like to thank my coauthors: Ramón Agüero Calvo, Josu Bilbao Ugalde, Fátima Fernández Pérez, Pablo Garrido Ortiz, Özgü Alay, Anna Brunström, Simone Ferlin and José Ramón Juárez Rodríguez. Their assessment, recommendations and contributions form the most important pillars on which this dissertation was built. I am especially grateful for their help in translating my natural way of expressing ideas into the common human language that everyone can understand. Although I never knew who peer-reviewed our publications, I would like to thank them too. Constructive criticism is critical to high-quality research.

I would like to thank Özgü Alay and her team for a fruitful stay in the University of Oslo. It was a really horizon-broadening experience. I would also like to express my gratitude to the people from other teams that I met there, especially Michael Welzl. Our conversations gave me a broader understanding of the Internet and its usage.

I would also like to thank the University of Cantabria, which I have already had the pleasure of thanking for awarding me the title of telecommunication engineer, and Ikerlan Technology Research Center, for offering me the opportunity to make my PhD thesis in such a diverse environment. Few PhD students are fortunate enough not only to engage in cutting-edge research but also to see the fruits of the research reach the marketplace.

My very special thanks to my mother. There are different stages of being stressed and nervous. She has seen all of the mine. Her continuous support is helping me these days as much as in childhood.

I would also like to thank my friends for staying close, supporting me and helping me to disconnect. The inspiration only comes to relaxed minds.

Last but not least, I want to thank Ukrainian people for showing the world that nothing is impossible. Ви неймовірні!

Resumen

Los usuarios de Internet buscan conexiones más rápidas y seguras, con una transferencia de datos fiable (integridad de la información). Hoy en día, los dispositivos se conectan a internet a través de enlaces inalámbricos susceptibles de sufrir interferencias y otros efectos no deseados, lo que puede generar pérdida de paquetes de datos. Otra característica de los usuarios de internet a día de hoy (2023) es que emplean dispositivos con múltiples interfaces de red, con acceso a diferentes tipos de redes. Por ejemplo, los smartphones pueden acceder a redes móviles y Wi-Fi. El uso simultáneo de múltiples interfaces de red, conocido como MultiPath (MP), podría aumentar enormemente la fiabilidad de las comunicaciones, duplicando datos en cada interfaz, o su capacidad, agregando los anchos de banda disponibles.

El protocolo extremo a extremo más popular, entre los que garantizan la integridad de los datos, Transmission Control Protocol (TCP), se diseñó para enlaces cableados. A pesar de que se hayan diseñado múltiples extensiones y mejoras para TCP, son difíciles de implementar por dos razones: TCP se implementa en el espacio kernel y los nodos intermedios de la red están sobre-optimizados para las cabeceras clásicas de TCP, descartando las que no sean conocidas. Así, los despliegues de actualizaciones de TCP requieren actualizar muchos dispositivos, por lo que se trata de una tarea que quizá nunca se complete del todo en la Internet moderna. A pesar de que existe un protocolo MP basado en TCP, MultiPath Transmission Control Protocol (MPTCP), este hereda el mismo problema de despliegue de nuevas versiones.

Con el fin de optimizar el tráfico web, Google Inc. comenzó el desarrollo de un nuevo protocolo fiable basado en User Datagram Protocol (UDP), QUIC. Su especificación final ha sido finalizada en el marco del Internet Engineering Task Force (IETF). QUIC se ejecuta en el espacio de usuario de los dispositivos que se encuentran en los extremos de una conexión, apareciendo ante los nodos de la red como el contenido de los datagramas del protocolo UDP. QUIC garantiza la integridad de los datos, reduce la latencia a través de diferentes estrategias y encripta sus paquetes, ofreciendo más seguridad a sus usuarios que TCP por sí solo. Aunque QUIC tiene un mecanismo más eficiente para recuperar paquetes perdidos, puede reducir todavía más el tiempo de recuperación utilizando la extensión Forward Error/Erasure Correction (FEC). La primera versión de QUIC no incluye una funcionalidad MP. Sin embargo, el gran interés en esta extensión está impulsando su borrador de especificación desarrollado por el IETF. A pesar de los múltiples estudios de FEC y MP aplicados a QUIC, existen muy pocos trabajos que hayan evaluado un uso conjunto de ambas extensiones en este protocolo.

Esta tesis se centra en mejorar aún más un protocolo de transporte que ya de por sí es eficiente con las funcionalidades FEC y MP. Para implementar cada extensión se revisa de manera profunda el estado del arte existente y se evalúan con detalle las soluciones propuestas y sus desarrollos, emulando distintos tipos de tráfico, así como diferentes escenarios de red. Además de comprobar la reducción del tiempo de

transmisión que ofrecen las técnicas propuestas, se estudian las interacciones entre las extensiones y el protocolo QUIC original. Esto es valioso para combinar múltiples extensiones, especialmente aquellas que puedan surgir en futuro alrededor de MP.

Abstract

Internet users are striving for faster and more secure connections, together with reliable data transfer (data integrity). Nowadays, end devices connect to the internet through wireless links, susceptible of interference and other hostile situations, which can end up in data packet erasures. Another characteristic of modern internet users is that they have use devices that entail multiple interfaces with access to different networks. For instance, smartphones can access cellular and Wi-Fi networks. The simultaneous use of multiple network interfaces, known as MultiPath (MP), could greatly increase the reliability by duplicating data, or the capacity, by aggregating the available bandwidths.

The most popular end-to-end protocol that ensures data integrity, Transmission Control Protocol (TCP), was designed for wired links. Although multiple extensions and improvements have been designed for TCP, these are hard to deploy for two reasons: TCP is implemented in kernel space and the intermediate nodes in the network are over-optimized considering the traditional TCP headers, discarding any unknown one. Deployments of TCP upgrades require updating many devices, a task that might never be fully complete in modern internet. Despite there is a TCP based MP protocol, MultiPath Transmission Control Protocol (MPTCP), it inherits the same problem of new version deployment.

In order to optimize web traffic, Google Inc. began the development of a new reliable transport protocol based on User Datagram Protocol (UDP), QUIC. Its final specification has been recently concluded by Internet Engineering Task Force (IETF). QUIC runs in user space of connection endpoints, while intermediate network nodes treat it as UDP payload. It ensures data integrity, reduces latencies by means of various mechanisms, and encrypts its packets, offering more security to its users than TCP alone. Although QUIC has a more efficient mechanism to recover lost packets, it can still reduce recovery time using a Forward Error/Erasure Correction (FEC) extension. The first version of QUIC does not include a MP functionality. However, the strong interest in this extension is fostering its draft specification, developed by IETF. Despite there exist multiple studies of FEC and MP applied to QUIC, very few works have tackled the evaluation of jointly using both extensions.

This thesis focuses on further improving an already efficient transport protocol with FEC and MP functionalities. To implement each extension we carefully review existing state of the art, and we thoroughly assess the performance of the proposed solutions, by considering different traffic patterns over various emulated network scenarios. Apart from verifying the transmission time reduction brought by the proposed techniques, we observe the interactions between them and the original QUIC protocol. This is of utter relevance when considering the combination of multiple extensions, especially those that affect the upcoming MP QUIC.

Contents

List of Figures

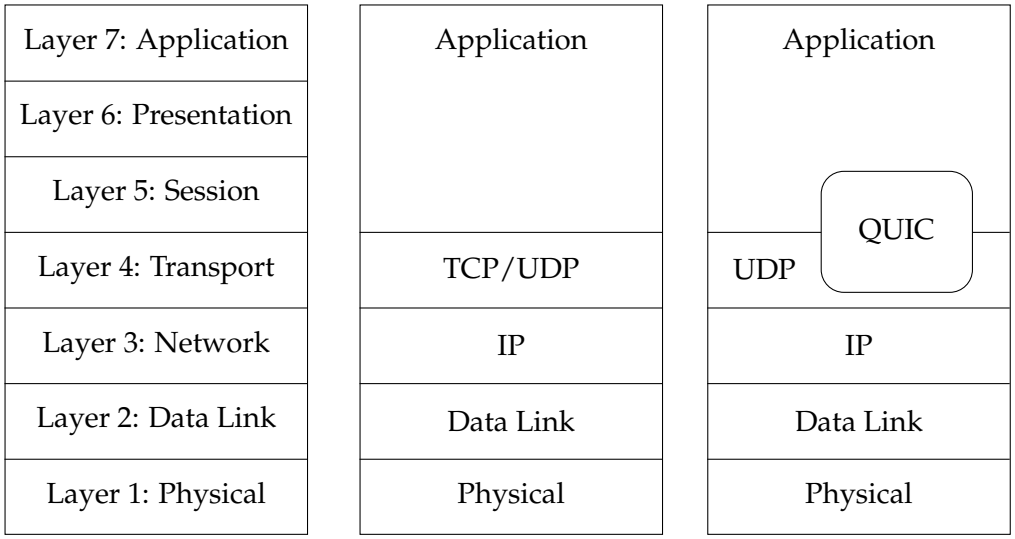
List of Abbreviations

1

Introduction

Internet is a global network interconnecting millions of users across the world. The growing demand for lower latency and greater Bandwidth (BW), stability and reliability has been pushing its evolution. Wireless links are used to offer mobility and deployment flexibility to end users. Optical fiber is used for wired links, greatly increasing the available BW and reducing propagation latencies. The devices connecting to internet have also evolved. Not only they gained computational power, but also mobility and the ability to connect to different kinds of networks. A typical smartphone has Wi-Fi, mobile, Bluetooth and Near-Field Communication (NFC) network interfaces. Protocol stacks also change to enable new features offered by modern networks and demanded by the users, such as using multiple network interfaces simultaneously. For instance, a smartphone could use both Wi-Fi and mobile connections, to either duplicate the data for a greater reliability or to aggregate the available BWs.

There is a vast number of communication protocols that can be used in different cases. Protocols are organized in layers, abstraction levels with specific functions. Figure 1.1 shows different views on protocol stack organization. Figure 1.1a represents the Open Systems Interconnection (OSI) model [II94] and Figure 1.1b, the legacy protocol suite specified in [Bra89a, Bra89b]. Data link layer protocols are chosen to adapt to the physical layer, which is chosen at device/network installation and never changes. Despite its stability over time, the variety of technologies used to interconnect devices is enormous: Ethernet, coaxial cable, optical fiber, Wi-Fi, ZigBee, Bluetooth, 3/4/5G, etc. Protocols from the layers above transport are chosen by the programmer of each application. The protocols present in almost every possible stack are found in network layer, IPv4 [Pos81a] and IPv6 [DH17], and transport layer, Transmission Control Protocol (TCP) [Pos81b, Edd22] and User Datagram Protocol (UDP) [Pos80]. The growing number of Internet of Things (IoT) devices use highly customized protocols in each



(a) OSI model [II94] (b) Classic internet protocol suite [Bra89a, Bra89b] (c) QUIC protocol stack [IT21]

Figure 1.1: Internet protocol stacks overview.

layer. Nevertheless, to exchange information with remote endpoints their messages will need to traverse networks with IP routing. Even if IoT devices in their local network use another protocol instead of IP, there will be a traffic aggregator with internet access translating packets to IP.

Given the few variability in network and transport layers, improvements with the biggest impact should be sought there. However, the protocols from these layers are usually implemented in kernel space of the end devices and the network nodes. Besides, the latter are optimized to work with the headers of the known protocols. Updates to TCP that change the packet header may result in certain nodes (middleboxes) discarding the updated packets as corrupt. QUIC has recently emerged as an alternative to widespread TCP. QUIC formerly stood for "Quick UDP Internet Connections" [HISW16], however, now QUIC is a name, not an acronym [IT21, Section 1.2, Paragraph 3]. First developed by Google for optimizing web traffic [LIB+17], its final design has been carried out by Internet Engineering Task Force (IETF), finishing the specification of the first version in 2021 [IT21, TT21, IS21]. It is a reliable transport layer protocol running on top of UDP, as shown on Figure 1.1c. Its deployment and updates are not dependent on network devices, but only on the endpoints, which implement it in user space. QUIC counts on version negotiation to enable coexistence of different QUIC versions as the protocol is upgraded over time [IT21, SR22]. Key players such as Google, Apple, Microsoft, and Facebook have developed their own QUIC implementations [Pau23], fostering its widespread adoption.

QUIC achieves lower latencies than TCP by shortening the handshake and improving loss recovery mechanism. A protocol with a faster loss recovery is better suited for communications carried through wireless links, which are susceptible of suffering interference. Packets with irreparable errors are discarded at data link layer, transport layer protocols only see packet erasures. Both QUIC and TCP employ Automatic Repeat reQuest/Query (ARQ) mechanisms to detect and recover lost packets. Recovering one packet with ARQ takes at least one Round Trip Time (RTT): the transmission in one way of the ACKnowledgement (ACK) signaling the loss, plus the transmission in another way of the lost packet. Nowadays internet users usually access internet through wireless links, using either Wi-Fi or mobile networks. IoT devices also exploit wireless links in most of the cases. Frequent packet erasures with delays of *at least* one RTT can significantly delay transmission completion. To further reduce recovery time, another technique can be applied: Forward Error/Erasure Correction (FEC). This technique combines source symbols (original packets) to build coded or repair symbols that will be sent to the network along with the source ones. Lost original packets can be recovered from the coded ones without any retransmission, achieving recovery delays below one RTT.

Another technique capable of significant transmission time reduction is MultiPath (MP), which consists in concurrent use of multiple network interfaces. MP can be used to increase reliability by duplicating data in each path and avoiding retransmissions, or to aggregate the available BWs, and finish the transmission earlier. There is a MP protocol based on TCP, MultiPath Transmission Control Protocol (MPTCP) [FRH+20], which is as hard to update as Single Path (SP) TCP. A MP extension for QUIC would inherit QUIC improvements and advantages, such as latency reduction and protocol versioning. The vivid interest for a MP extension for QUIC protocol has driven multiple IETF draft specifications, already unified in a unique proposal [LMC+23].

Upgrading a widely used transport layer protocol, namely TCP, would theoretically improve most of communications. However, given the difficulties to deploy TCP upgrades, new extensions might never be used outside their testbeds. QUIC is a new transport protocol whose deployment requires no change in network nodes, which see this protocol as UDP payloads. QUIC operates in user space, and it includes version management, making new extensions easy to implement and deploy. Although it was designed for web traffic, given that it implements the same functionality as TCP, QUIC can be used for other types of traffic. QUIC includes mechanisms that help reducing latency, however, other mechanisms could allow reducing the latency even further.

1.1 | Motivation and Objectives

Improving communications usually translates in ensuring data delivery integrity, reducing transmission time and providing high security (data privacy). The first two are the most easily appreciated improvements. TCP alone ensures data integrity. QUIC, on top of that, reduces latencies compared to TCP, and establishes the use of TLS 1.3 encryption in its specification [TT21]. Working with QUIC seems an obvious step to take towards a general communication improvement.

QUIC recovers its losses faster than TCP, which is especially beneficial for transmissions traversing error-prone wireless links, thus reducing the overall transmission time as well. This time can be further reduced with the use of different extensions. QUIC extensions are easy to deploy, thanks to QUIC running in user space of the endpoints using it. FEC can reduce recovery time and MP can greatly reduce the overall transmission time. Combining both techniques might result in achieving the same level of redundancy as by duplicating data in MP, making a much better use of the total available BW.

The main goal of this PhD thesis is **to minimize latency in end-to-end communications**. To achieve that, we extend QUIC protocol with FEC and MP techniques. This overall goal boils down to the following specific objectives:

- *Forward Error/Erasure Correction (FEC)*
  Increase reliability and reduce latency by means of coding techniques.
  - Identify the coding schemes that could be used with QUIC, which already includes a reliable delivery mechanism (ARQ).
  - Analyze the interactions required between FEC extension and the base protocol.
  - Design and implement a QUIC extension to protect the end-to-end traffic with coding techniques. The extension should be configurable and extendable with new coding schemes.
  - Include a mechanism to adapt the generation of redundancies to the observed network conditions.
  - Evaluate the resulting extension in different environments, using different traffic patterns and services.
- *MultiPath (MP)*
  Reduce latency by aggregating BWs available through different network interfaces.

  - Analyze the interactions required between MP extension and the base protocol.
  - Add MP extension to QUIC. IETF draft specifications of MP will be followed, but not strictly. MP design could be adjusted for the sake of optimal functionality and latency minimization. At the beginning of this thesis the only MP document at IETF was [CB18], based on the work presented in [DCB17]. The active MP draft at the time of writing is [LMC+23].
  - Ensure the proposed extension is working accordingly. As we thoroughly explain in Chapters 2 and 4, poor design of path selection algorithms, ignoring possible receive buffer limitations, can lead to performance degradation, resulting in greater than SP QUIC transmission times.
  - Evaluate MP extension in different scenarios using different types of traffic.

- *FEC and MP*
  Optimize the joint use of FEC and MP extensions.
  - Analyze the interactions required between both extensions for a combined use. Review whether any of the extensions needs any update to their interactions with the original QUIC protocol.
  - Update both extensions for a joint operation.
  - Evaluate the joint benefit of using both extensions separately.
  - Evaluate the benefits of using both extensions jointly.
  - Compare evaluation results, assessing whether there is any improvement in a joint use over the separate use of both extensions.

1.2 | Document Structure

To tackle the objectives set for this thesis, we structure the rest of the document as follows:

- In Chapter 2 we review the state of the art. First, we focus on QUIC, discussing its features and functionality. Next we review the available coding techniques that could be used in our FEC implementation. Then, we review existing uses of FEC in transport layer protocols. We conclude the chapter with an overview of MP protocols, paying special attention to path scheduling and the combination of MP and FEC.

- Chapter 3 describes our FEC extension for QUIC. We start by defining the terminology and coding techniques considered for the extension. Next we select a coding scheme that we expect to be efficient and evaluate its performance under different configurations. Then we implement our FEC extension for QUIC, which we call rQUIC after the previous work in this direction [GSF+19]. We describe rQUIC architecture and the adaptive coding scheme chosen to protect QUIC connection. We thoroughly evaluate the implementation in different simulated networks with different traffic patterns.

- Chapter 4 describes our MP extension. We start by identifying path specific and data related components of QUIC, to replicate path specif structure for each path. To enable synchronization between paths, we review the necessary changes in different QUIC components, defining the new ones. We pay a special attention to path scheduling. Unlike TCP and MPTCP, QUIC can have multiple data sources represented by streams, a data organization abstraction used in HyperText Transfer Protocol version 2 (HTTP/2). We propose a solution to handle data from different streams over multiple paths. We conclude the chapter with a careful review of our FEC design, updating it to include new advances on FEC and enable a joint operation with MP.

- In Chapter 5 we summarize the work developed in this thesis, drawing conclusions and identifying future research lines.

2

Background & Literature Overview

As networks and the traffic traversing them evolve, the need to evolve the communication protocols as well becomes a goal. Intermediate nodes in different networks are optimized to work with the most popular reliable transport protocol, TCP. The introduction of new features and improvements in TCP becomes a hard challenge. On one hand, TCP is implemented at kernel. On the other hand, the intermediate nodes are expecting to see packets with the TCP header they know, and if the header includes new signaling, some of the intermediate nodes could discard such packets as corrupt.

QUIC is a new transport protocol implemented in user-space to be easy to update. It encrypts most of its fields to avoid intermediate nodes interfering with its evolution [LIB+17]. Among other things, QUIC reduces latencies in different ways. In this chapter we review QUIC protocol as well as the techniques that can be used to further reduce latency. More specifically, we focus on FEC and MP extensions. We discuss some of the most relevant related works that proposed a combination of QUIC and these extensions. Despite QUIC being a very novel protocol, there already exist many works extending its functionality. Nevertheless, we still need to recur to the extensions of other protocols, especially TCP, where both FEC and MP have been implemented with different variations and extensively evaluated.

2.1 | QUIC

QUIC is a connection-oriented transport protocol that follows the traditional client–server architecture. QUIC was initially proposed by Google [LIB+17] and its first version was recently standardized by the IETF [IT21, TT21, IS21]. While retaining some of the key TCP features, such as reliable delivery and Congestion Control (CC), QUIC is built on top of UDP, with the aim of solving the challenges associated with protocols run-

ning on top of TCP, such as connection-setup time, Head of Line Blocking (HoLB), and middlebox interference.

QUIC establishes a secure connection with TLS 1.3 in just one RTT, since its own and TLS handshakes are run in parallel. Thanks to TLS 1.3, application data can be sent in zero RTT in subsequent reconnection.

Following HTTP/2 web object organization, QUIC implements streams, a data handling abstraction that can be defined as sequences of ordered bytes. This feature not only helps to manage data, but it also reduces the impact of HoLB. When one data segment is lost or arrives out of order, the subsequent segments are retained in the receive buffer without being delivered to the application till the missing one arrives. While in TCP the HoLB affects the whole connection, in QUIC it hinders only one of the streams multiplexed within a connection.

Furthermore, QUIC brings additional latency reduction, thanks to its loss detection mechanisms, including "Early Retransmits" and tail loss probes. QUIC introduces significant improvements compared to traditional reliable transport layer solutions. Some benefits of QUIC loss recovery mechanisms are a consequence of the numbering scheme, where packet identifiers are never repeated. If a packet is lost, QUIC retransmits the same information in another packet, with a different packet number, removing uncertainty about the actual packet that is confirmed when an ACK is received. As a result, QUIC achieves more accurate RTT measurements and can identify spurious retransmissions [IS21].

QUIC loss detection algorithms are ACK-based, with a probe timeout to ensure that the ACKs are received. Packets are considered lost if one of the following conditions is met:

- A packet was sent before an acknowledged packet and it has not been acknowledged, and either of the following events occurred: (i) the packet was sent with a packet number three times smaller than the latest acknowledged; or (ii) it was sent after 9/8 RTT expiry time.

- The probe timeout expires, that is, the last packet was sent, and the corresponding ACK has not arrived in the expected time, calculated as the sum of a smoothed estimation of RTT, the variation of its estimate and the maximum ACK delay, which is the time that an endpoint can introduce before sending an ACK, a parameter negotiated at handshake. Upon reaching this timeout, the endpoint sends the last packets without modifying their content to either recover the loss, or to at least trigger an ACK transmission. This timeout is duplicated upon every probe transmission.

Middleboxes are networking elements that manage TCP traffic, analyze TCP segments and modify them with the aim of leveraging an optimal performance [HNR+11]. This might hinder TCP enhancements, as existing middleboxes would require to be updated if there is any TCP modification. On the other hand, if TCP were upgraded, this could affect several nodes and devices, because its implementation is usually at the kernel of operating systems [LIB+17]. As for middleboxes meddling the connection, QUIC encrypts its packets, leaving only unprotected header fields that are required to identify a QUIC connection at an endpoint [TT21]. In addition to security reasons, this encryption is used to avoid any interaction with middleboxes, and thus, protocol ossification [LIB+17]. Since QUIC packets are processed as opaque UDP payloads, middleboxes are not aware of them, ensuring a smooth interaction and facilitating migration and update strategies. QUIC was designed to be deployed in the user-space, to enable the management of computational resources with other applications within the same node, or the establishment of logging levels as needed. In any case, QUIC can also be implemented in the kernel [WBRP18], with the main goal of enhancing its performance.

Although QUIC facilitates the integration of new extensions, not all endpoints might be able to upgrade at the same time, resulting in the coexistence of several QUIC versions. The version negotiation mechanism promoted by QUIC ensures an appropriate operation, by allowing devices to negotiate the version they will use in the connection during the establishment phase [IT21, SR22]. In addition to the simplification of the protocol updates, this mechanism also enables the possibility of extending QUIC with new functionalities, which devices may even share as plugins [DCMP+19].

Next, we introduce FEC and Network Coding (NC) basic functionality, focusing on the features that will be considered in our design.

2.2 | FEC and Network Coding

Traditionally, there are two main causes of information loss events during communication: network congestion and transmission errors. Reliable protocols that ensure data integrity, such as TCP [Pos81b] and QUIC [IS21], recover lost information using ARQ mechanisms. These consist of signaling lost information and taking care of its retransmission. On the other hand, FEC [ACC+07] techniques promote a different approach, allowing the recovery of lost chunks of information or source symbols, by sending redundant symbols along with the source ones. Redundant symbols are built as combinations of the source ones, which is why they are also called coded or repair symbols. FEC can improve latency, as losses might be recovered without additional signaling,

and thus there is no need to wait for a retransmission, at the expense of increasing the overhead sent to the network.

It is worth noting that depending on coding scheme implementation, a transport protocol's data packet could fit multiple symbols, both source and coded ones. In this case, a packet loss implies the loss of multiple symbols, complicating loss recovery at the decoder. To make the most of coding techniques, this thesis considers transmissions of one symbol per packet. From this moment on the meaning of 'packet' is equivalent to 'symbol'.

FEC acts only at communication endpoints, at the transmitter (encoder) and the receiver (decoder). However, at transport layer a communication link covers multiple networks in which multiple nodes forward the information towards its destination. Ahlswede *et al.*. introduced the concept of Network Coding (NC), where intermediate nodes can perform coding operations on ongoing communication flows [ANLY00]. The interaction with the information flow offers different opportunities to optimize network usage. Depending on the kind of optimizations, NC can be of three types: analog, inter-session and intra-session [KKR12].

- *Analog* or *Physical-layer* NC is applied at physical layer. In the case of two endpoints in a shared access medium sending messages to each other through an intermediate relay node, there will be a total of four transmissions: one for each sender and two for the relay node. In physical NC the messages can be sent simultaneously, interfering with one another. The relay node receives the interfered message, amplifies it, and sends it back to the network. The endpoints can extract the new message from the interfered one with the message they sent. Thus, the simultaneous endpoints' transmissions and the relay retransmission reduce medium access time by half.

- *Inter-session* or *Inter-flow* NC combines data sequences from different data flows to improve network efficiency. In the previous example, the endpoints cannot send their messages interfering with one another. However, the relay node can combine both messages with an XOR operation and send the combination instead of the two original messages. The endpoints can recover the incoming message from the combined one using the message they sent.

- *Intra-session* or *Intra-flow* NC applies network coding techniques within a data flow. It can be viewed as an FEC operating over each hop on the communication path (between the two endpoints), with NC-enabled intermediate nodes. Note

that the NC headers of packets protected with this technique must be visible to the intermediate nodes.

Ho *et al.* proposed Random Linear Network Coding (RLNC) [HKM+03, HMK+06], a coding technique where all the transmissions correspond to a random combination of source symbols. These are grouped into data blocks or generations of N symbols and combined to create $N + K$ coded symbols. Whenever the destination (or an intermediate node) receives at least N linearly independent coded symbols, the original information can be recovered. Because the combinations are randomly built, some useless transmissions may occur (not linearly independent symbols). Hence, additional transmissions may be needed. The linear combination of coded symbols can also be seen as a linear combination of source symbols; thus, intermediate nodes can perform recoding operations without needing to decode the entire generation. This has proven to yield some benefits under certain circumstances, but if these are not met, and intermediate nodes simply forward the received packets towards the destination, RLNC boils down to Random Linear Coding (RLC), where only the source node performs coding operations.

The computational complexity of RLNC may be unfeasible for constrained devices. Wang *et al.* have shown that the use of sparse coding matrices, where some of the coding coefficients are zero, can alleviate this problem [WL06]. Hence, when building a coded symbol not all the source symbols are considered. A particular case of sparse NC is a systematic coding scheme in which source symbols are sent to the network without coding, together with additional coded symbols, to compensate for eventual network losses [HPFL09]. In this context, the coded symbols can be referred to as redundant symbols. Systematic coding simplifies not only encoding, but also decoding operations, because source symbols can be used as soon as they are received, and this can reduce the overall latency.

As mentioned earlier, RLNC groups source symbols into blocks or generations. Another approach to building coding symbols, known as convolutional coding [LMZG97], is protecting source symbols from a sliding window. Here, the equivalent of RLNC generations are overlapped, which brings two advantages: source symbols are protected by more coded ones; coded symbols are inserted more frequently, giving the decoder more opportunities to recover losses. A systematic RLNC with overlapped generations was discussed in [WGP+17] and further extended with ARQ in [GWP+18]. Both works considered a constant sliding window to select source symbols to build a generation. The frequency at which coded symbols were inserted between the source symbols, the code rate, was also constant, as well as the redundant symbols per generation. Given that

the coding window in both works was a multiple of the code rate, all source symbols belong to the same number of generations, and they are protected by the same number of coded symbols. This offers a different view on convolutional codes, where when one of the concurrent generations is finished, it is replaced with a new one. This view is exploited in this thesis (c.f.Section 3.2), focusing on concurrent generations rather than on a coding window.

A convolutional coding scheme that overlaps generations yields an overall latency reduction [WGP+17]. Because each source symbol is protected by multiple coded symbols, this type of coding offers more robustness to bursty losses. However, if bursts are long, a different coding approach might be necessary, such as interleaving [HWCK07], where generations are grouped in interleaving blocks. Rather than sending symbols from the same generation, all the i^{th} symbols from each generation are transmitted. Hence, consecutively transmitted symbols do not actually correspond to consecutive source symbols, thus distributing the impact of burst losses among all generations in the interleaving block.

Building interleaving blocks requires buffering all generations, which may cause additional latency. Stolpmann *et al.* suggested a different, systematic approach to interleaving network coded symbols, Interleaving with On-the-fly Coding [SPETG18]. Instead of waiting for completing an interleaving block, each new source symbol is covered by a different generation from the interleaving block under construction. In this way, all source symbols can be transmitted in order, as soon as they are generated, and interleaving properties are not lost, because burst losses are still distributed across multiple generations.

2.3 | Coding at Transport Layer

Various studies have exploited coding techniques to increase the performance (mostly throughput) offered by the transport layer, both for reliable-service (TCP) and real-time applications (based on UDP). Below, some of the most important examples are reviewed.

There is a lot of research focused on enhancing TCP, the most popular reliable transport protocol. In what follows we review some of them. FEC as an extension of a reliable transport protocol has been widely studied, which given TCP overwhelming adoption means numerous studies on the integration of FEC with TCP. Subramanian *et al.* have designed LT-TCP, a TCP modification that uses both ARQ and FEC to recover lost packets [SKR06]. This modification uses Explicit Congestion Notification (ECN)

signals to detect congestion events. Tsugawa *et al.* proposed TCP-AFEC, an adaptive coding scheme that inserts enough redundant packets to ensure the application perceived BW required for video streaming [TFH+07]. TCP-AFEC hides recovered losses from TCP CC. The authors argue that congested networks' collapse is avoided by traditional TCP mechanism when FEC fails to recover a packet and the loss is detected. Teshima *et al.* have proved that TCP-AFEC is not optimized for wireless LAN, and proposed TCP-TFEC [TOHI17], an adaptive coding scheme that is more suitable for this scenario. Krishnaprasad *et al.* developed another TCP modification for improving its performance over wireless links, TCP Kay [KTK15]. Apart from FEC operations, it increases ACK frequency to grow the congestion window faster after a loss event. An adaptive XOR FEC extension for MPTCP is presented in [FKCA18]. FEC efficiency is optimized with congestion window information, however, no change is done to CC, recovered packets mask the loss event. A more recent TCP modification is presented in [SKI19], where Sato *et al.* introduce an adaptive XOR coding scheme. As [SKR06], this scheme relies on ECN for congestion detection. In order to prevent lost packets' retransmission, Sato's FEC suppresses duplicate ACK packets, unless the decoder cannot recover the losses. Additionally, this FEC implementation applies on-the-fly interleaving similar to [SPETG18] with a variable interleaving block size.

A tight integration of FEC with a transport protocol is not the only approach to use FEC above link layer. Bolot *et al.* apply FEC to audio stream at application level in a TCP-friendly way, respecting congestion avoidance mechanism [BFPT99]. In [WCWC17] Wu *et al.* present a coding scheme for video transmission over TCP. Their goal is not only to shorten loss recovery, but also to address the poor performance of CC mechanism over wireless, and typically error prone, links. On one hand, this approach does not break the underlying transport protocol, but on the other hand, it is not preventing the unnecessary congestion window reduction induced by a non-congestion loss.

The use of FEC as an extension of a reliable transport protocol (mostly TCP) has been widely studied, in two main directions: (i) protecting TCP data flows, and (ii) extending TCP itself. An example of the first group is found in [WCWC17], where the authors optimized video transmission by recovering losses using an adaptive coding scheme. On the one hand, this approach does not break the underlying transport protocol, but on the other hand, it does not prevent unnecessary congestion window reduction that might be induced by non-congestion losses, which are very frequent in wireless links. Tsugawa *et al.* proposed TCP-AFEC [TFH+07], which is an extension of TCP with an adaptive coding scheme to improve video streaming. TCP-AFEC hides the recovered losses from TCP CC. The authors argue that the collapse of congested networks is avoided by the traditional TCP mechanism when FEC fails to recover a packet and the loss is detected.

Later, Teshima *et al.* proved in [TOHI17] that TCP-AFEC is not optimized for WLAN, proposing TCP-TFEC, an adaptive coding scheme that is more suitable for this technology. Sato *et al.* extended TCP using the XOR coding scheme in [SKI19]. They interleave generations, as was also suggested by [SPETG18], to mitigate burst losses and suppress duplicate ACK packets, thus allowing the FEC scheme to recover losses by itself. Packet losses are thus intentionally hidden from CC mechanisms, allowing high goodput to be maintained. Congestion avoidance relies on Explicit Congestion Notification (ECN) signals.

However, losses at transport layer are frequently caused by network congestion events. When coding techniques are applied to reliable transport protocols with CC, it is unclear how the latter should interact with the encoder. It is worth noting that there is an Internet Research Task Force (IRTF) draft describing these interactions [KLMW22]. When coding techniques are applied to a reliable transport protocol's data flow, that is, above the transport layer as in [WCWC17], coded symbols and packet losses are visible to the CC. However, when coding techniques are integrated within the transport protocol, successful recoveries may hide congestion losses from the Congestion Control Algorithm (CCA). Depending on the coding scheme, the impact on a congested network may not be negligible. Hence, an appropriate CCA might need to be aware of coding operations. To do so, the CCA should be able to distinguish between congestion and random losses, with the latter being responsible for unnecessary Congestion Window (CWND) reduction. The challenge of distinguishing the nature of packet losses has received significant attention from the scientific community. Truchly et at. review in [TSR19] some of the most relevant CCAs that have been proposed.

UDP datagrams protection with FEC has also been widely studied. Research on FEC protection of UDP data flows focuses on real-time multimedia streaming. For this purpose, the most common protocol is Real-time Transport Protocol (RTP) [SCFJ03] over UDP. For this reason, although not strictly required [SCFJ03], the most common underlying protocol is UDP. FEC protection can recover lost information with sufficiently low latency, which is why it can be used to successfully increase transmission reliability. RTP specifies different FEC extensions, such as the generic FEC specification [Li07] and XOR coding with interleaving [Beg10], the latter being extensively evaluated in [LD12]. Another example of UDP video streams protected with an FEC is [GNPS19]. The authors propose the simultaneous use of TCP and UDP over different network interfaces for smooth video streaming over HTTP, proving that their approach is efficient not only in terms of goodput, but also in terms of the energy consumption of mobile devices.

Another interesting approach that has recently gained relevance is extending QUIC with coding techniques, although there are not many evaluations of this approach,

mostly due to QUIC novelty. The first experiments integrating QUIC and FEC, carried out by Google, are described in [Swe16]. Although the results did not reflect a significant improvement, other researchers continued to work on coding techniques for QUIC. As a result, it has been contemplated in the corresponding standardization efforts, and the IRTF draft [SMRM20] focuses on how to implement a generic FEC scheme, while [RMSM20] focuses specifically on RLC. Furthermore, Michel *et al.* presented an FEC extension with a fixed code rate, assessing its performance over constrained links [MDB19]. This extension is subsequently converted into a portable plugin, integrated along with multipath and other features in the *pluginized* QUIC [DCMP+19]. Garrido *et al.* presented the first version of rQUIC [GSF+19], an extension of QUIC with an adaptive FEC. This thesis continues those experiments, extending for that purpose the original implementation with a more generic scheme, which might be configured to feature different coding solutions. As discussed in Section 3.6, the results demonstrate that this renewed implementation yields a better performance than the original one. In addition, the behavior that rQUIC exhibits when used with services having real-time requirements is also assessed, such as video streaming.

As introduced previously, intra-flow NC can be seen as an FEC scheme between coding capable nodes, applied on all hops along the path between the two endpoints. One of its more relevant benefits is the reduction of overhead introduced by coded packets, as more reliable hops could require fewer redundancies. CTCP achieves a greatly improved goodput compared to TCP both in controlled and real world scenarios. The authors of [KCP+13] proposed the use of NC in the transport layer. They argue that this approach grants backward compatibility with legacy network equipment, as link layer protocols are not changed. This can be seen as another advantage of extending TCP with NC, while it is even more important for QUIC, as it is meant to be implemented in the application layer. However, to the best of our knowledge, there is no proposal to extend QUIC with NC, despite the benefits reported for TCP [KCP+13, SSM+11, RE16, APA19]. This work argues that one of the main obstacles lies in QUIC philosophy, since it prevents a tight interaction with middleboxes. If no interaction with middleboxes is allowed, the intra-flow NC would boil down to the traditional FEC. In any case, the design and implementation of rQUIC described here is conceived to be able to integrate the main functionalities of NC.

2.4 | Multipath – Multiple Network Interfaces

Nowadays, the devices connecting to internet have multiple network interfaces. In smartphones both Wi-Fi and mobile networks are used to exchange information. If mobile traffic tariff is not restrictive, both interfaces could be used simultaneously to finish any data exchange earlier. The use of multiple transport layer paths or MultiPath (MP) is not a new concept. MP has been implemented as new protocols and extensions to the existing ones on all layers of the internet stack. There are multiple solutions to aggregate the available BWs in link and network layers [LLO+16]. However, implementations at transport layer can have more information for a better path scheduling. A reliable transport protocol, such as QUIC, TCP or Stream Control Transmission Protocol (SCTP), can estimate the latency, the available BW and the loss rate between both ends of a communication. In [LLO+16] Li *et al.* review and group transport protocols in two main groups: TCP based and SCTP based. The only exception is MP Datagram Congestion Control Protocol (DCCP), an extension of an unreliable protocol for video streaming. SCTP is message oriented partially reliable transport protocol designed with multi-homing support [STkN22]. QUIC with proper extensions could cover the functionality of TCP, SCTP and DCCP, including the respective MP extensions.

Despite QUIC novelty, there have been multiple studies of MP extension in this protocol. In 2017 DeConinck *et al.* published their MP QUIC design and evaluation [DCB17], which was used to prepare the first IETF specification draft for MP extension [DB21]. Among the available plugins were MP and FEC extensions. In 2018 Viernickel *et al.* published a similar design [VFR+18]. In 2019 DeConinck *et al.* presented a new QUIC extension that supported plugins [DCMP+19]. It enables the endpoints sharing any necessary extension between each other. Liu *et al.* published another draft specification for MP QUIC, which was used to build the implementation called XLINK. It was used for real-life video streaming. A detailed analysis of XLINK performance was published by Zheng *et al.* in [ZML+21]. Christian Huitema also published a draft specification of a simple MP QUIC extension. Later, the three draft specifications [DB21, Hui21b, LMH+21] have been unified into one specification proposal, which at the time of writing has been updated to the version 04 [LMC+23].

One of the biggest challenges for any MP implementation is scheduling data packets for each path. The rest of this section focuses on path scheduling and the combination of MP and coding techniques.

2.4.1 | Schedulers

The main goal of using different paths simultaneously is to maximize the throughput and to minimize the latencies. To achieve this, MPTCP default scheduler chooses the path with minimum RTT as long as it is available [FRH+20]. However, relying only on RTT is not enough for an optimum scheduling. It is also important to evaluate network losses and transmission rate on each path, as discussed in [NXHS14, KLL17].

To prevent HoLB at receive buffer caused by out-of-order arrivals, which is especially critical in heterogeneous paths, some scheduling algorithms estimate the arrival time for the packets each path and alter transmission order of the data packets to optimize in-order arrivals [SBL+13, KLM+14, SCW+18]. Yang *et al.* present a slightly different scheduler: instead of scheduling multiple packets in a given time interval, it schedules only 1 packet [YWA14].

Given the fundamental importance of receiving packets in the correct order in TCP, several studies have focused on minimizing the delay caused by HoLB [FAMB16]. Ferlin *et al.* proposed BLocking ESTimation (BLEST) algorithm, which schedules packets based on HoLB predictions. Choi *et al.* use the estimated BWs to avoid HoLB [CCA+17]. Their Optimal Load Balancing (OLB) scheduling uses a weighted round-robin (WRR) that assigns packets to each path according to a given weight, which can achieve optimal performance.

Taking excessive care of in-order delivery can result in aggreagted BW underutilization. To avoid that, Lim *et al.* evaluate subflow RTT, BW, and queued data in the send buffer to calculate the transmission completion time, choosing a path for a new packet based on Earliest Completion First (ECF) principle [LNTG17]. Decoupled Multipath Scheduler (DEMS) [GNM+17] minimize download completion times by measuring the data chunk length and sending the first packet forward on one path and the last packet back to the start on another path until there are no more packets to transmit.

Another way to assign traffic to a specific path is to consider the application layer's requirements. Fahmi *et al.* implement Stochastic Object-aware Scheduler (SOS) to reduce the delay of downloading application objects [FLKC18]. Instead of measuring the packet delay, SOS measures the delay of the object and schedules the object to the proper path. Shreedhar *et al.* proposed another scheduler based on a cross-layer approach, QAware [SMKK18]. It combines network information as the delay and the local queue buffer occupancy getting a big aggregate throughput.

Other scheduling algorithms assume a sufficiently large receive buffer. This assumption is frequently met in web traffic optimization, carried by HTTP/2 streams, which are mapped to QUIC streams if QUIC is used. Viernickel *et al.* provided a stream-to-path

scheduling policy based on MPTCP default minimum-RTT scheduler [VFR+18]. Rabitsch *et al.* proposed a stream-aware scheduling for MP QUIC [RHB18]. They extend the ECF scheduler from [LNTG17] to avoid a stream completion delay by a slower path and to allocate the fair share of aggregated BW based on stream priority. In 2019 Shi *et al.* proposed a different stream-aware scheduler: each stream is sent through only one path [SWZL19]. Top priority stream is assigned to the path where it would complete first. If various streams share one path, the BW assignment is made regarding their priorities.

A MP protocol could be used by a mobile device whose paths' parameters are heterogeneous and very dynamic. To properly adapt to such scenarios, Wu *et al.* propose a reinforcement learning scheduling algorithm Peekaboo, which they implement in MP QUIC [WAB+20]. Later a version optimized for 5G networks has been presented [WCF+21].

Wang *et al.* designed a MP scheduler for HTTP/2 web traffic [WGX19]. Given that streams represent web objects and none of them can be used till its transmission is complete, the authors send only one stream at a time ordered by priorities provided by HTTP/2. To ensure the earliest completion of each stream, they calculate stream data distribution among the paths based on RTT and BW estimations and the queuing time of the stream. In 2020 Shi *et al.* presented another scheduler focused on minimizing transmission completion time. Instead of solving a system of equations to determine the share of stream data to send through each path, they schedule packets based on stream priorities till the delivery time for the last scheduled packets is practically the same for all paths. Then the next packets are distributed among paths following BW estimation and stream priorities.

2.4.2 | Multipath and Coding Techniques – Joint Use

Most of existing research about the combination of MP and coding techniques are based on MPTCP protocol. MP Loss-Tolerant (MPLOT) applies a dynamic FEC scheme to all paths, improving the aggregated throughput compared to MPTCP [SKK+08]. Li *et al.* present a NC based MPTCP (NC-MPTCP), which applies NC to some of the paths. Some paths deliver the original data, while others, the coded packets [LLC12]. Coded TCP (C-TCP) applies systematic block codes with a modified CC [KPUM12]. The new CC reacts not only to losses, but also to delays, and replaces the CWND concept with tokens which authorize the sender to transmit more data. Li *et al.* present an adaptive systematic coding MPTCP which is focused on mitigating packet reordering at the receive buffer rather than on packet erasure correction [LLT+13]. Fountain code-based

MPTCP (FMTCP) aims at mitigating path heterogeneity by sending redundant packets on different paths generated with fountain codes [CWW+15]. QuAlity-Driven MultI-path TCP (ADMIT) uses FEC to optimize real-time high definition video [WYC+15]. Stochastic Earliest Delivery Path First (S-EDPF) schedules packets and applies FEC by considering the stochastic time-varying nature of path delays [GSKL17]. Ferlin *et al.* use adaptive XOR coding in MPTCP [FKCA18].

MP and FEC applied to UDP is mostly used for multimedia transmission. Chow *et al.* build their own MP UDP with FEC for video streaming [CYX+09]. Kwon *et al.* use both TCP and UDP for multimedia transmission with systematic Raptor codes protection [KGPS14]. Control traffic is sent through TCP connection and data, through UDP. Wu *et al.* combine TCP and UDP for a BW-efficient video streaming considering different quality metrics [WYC+16]. This work also uses systematic Raptor codes. Gabriel *et al.* use NC techniques over MP UDP for a more general case of time-critical tactile internet [GAF18].

Due to QUIC novelty, there are few works evaluationg the joint use of MP and FEC extensions. A QUIC implementation with FEC and MP extensions has been presented in [DCMP+19]. However, FEC and MP are implemented as separate extensions. To the best of our knowledge, the first QUIC extended with FEC and MP that combines both to maximize the benefits is [VW21]. Applying XOR coding and sending coded packets only on the slowest path (unless paths are severely congested or other performance degradation are found) they achieve significant reduction of transmission completion time.

Apart from TCP and UDP based solutions, other protocols are also proposed. In 2019 Chiariotti *et al.* introduced Latency-controlled End-to-end Aggregation Protocol (LEAP) [CKZC19]. LEAP optimizes the joint work of CCA, ARQ, FEC and MP scheduling using custom solutions for each module.

3

Forward Error Correction for Transport Protocols

This chapter covers the use of FEC with a low latency protocol, namely QUIC. Before extending QUIC with FEC, an analysis is done to understand which coding schemes would be good for low latency communications, and more specifically, for QUIC. This analysis includes a deeper view on a convolutional systematic coding scheme that uniformly protects all source symbols with a uniform distribution of the coded ones.

Next, FEC extension for QUIC is described, its architecture, operation and implementation details. The combination of QUIC and FEC is given the name 'rQUIC' (*r*obust QUIC), continuing with the efforts started by Pablo Garrido [GSF+19, Gar18]. rQUIC is thoroughly evaluated with simulation campaigns, focusing on representative traffic patterns over networks with packet erasures of different degrees.

QUIC specification finished in 2021, after we finished the design and evaluation of our FEC extension. Throughout this chapter we reference QUIC specification drafts on which the original QUIC code was based rather than the Request For Comments (RFC) documents.

3.1 | Coding Variants and Characteristics

This section covers coding features that have been particularly considered for this work. It also introduces naming conventions we will use throughout the document since the terminology used in rQUIC development slightly differs from the one recommended in [AAB+18].

One such difference is the use of the *generation* concept. Coding operations were originally intended for RLNC operation. Hence, the term *generation* is widely used, de-

fined here as a group of source symbols used to build coded symbols. However, as will be explained later, the rQUIC design discussed here aims at systematic convolutional coding schemes. A more appropriate term in this context is *the encoding window* [AAB+18].

Furthermore, the *code rate* is defined, from an implementation point of view, as in [GSF+19]: the rate between source and coded symbols, which is different from the definition in [AAB+18]. Thus, the code rate used in this study represents the number of source symbols after which a coded symbol should be inserted. If only one coded symbol is built for each generation, then the code rate equals the generation size.

An FEC implementation that is unaware of the transport layer protocol might send multiple symbols per packet. However, when losses occur at the transport layer they affect entire packets, not just a portion of them. Splitting a packet into multiple symbols implies that redundancies that are required to successfully decode a generation can be split between two or more packets. With a code rate perfectly adjusted to the network losses, a lost packet with both source and coded symbols might never be recovered. As introduced in Section 2.2, the symbols we consider comprise the whole packet, which is why the words 'symbol' and 'packet' are used interchangeably. In addition, coded packets can be referred to as redundant packets or redundancies.

As already mentioned in Section 2.2, convolutional coding with a constant coding window, a multiple of the code rate, leads to uniformly overlapped generations, where each source symbol belongs to the same number of generations. We define the *overlap* as the number of generations to which any source symbol belongs. An overlap of 1 corresponds to the traditional block coding approach. When using systematic convolutional coding, along with a code rate that adapts to losses, focusing on overlap rather than on encoding window might help to achieve a more uniform protection of source symbols, and a more uniform distribution of coded symbols among the source symbols. A more detailed study of the overlapping scheme is presented in Section 3.2.

As introduced in Section 2.1, QUIC advanced recovery schemes significantly reduce its recovery time compared to TCP. Some coding schemes have an intrinsic delay that may be longer than the original ARQ recovery. This FEC implementation is intended to be an extension, not a replacement for QUIC ARQ mechanism.

Following are summarized the coding approaches considered in rQUIC design.

- Systematic coding. Correctly received source packets can be used immediately without waiting for the entire generation.
- Adaptive coding. To minimize the overhead introduced by coded packets, the number of redundancies is adjusted to the losses observed in the network.

- Convolutional coding. The overhead introduced by coded packets can be distributed more uniformly by focusing on overlapping the generations, thus reducing the waiting time between recovery attempts at the decoder.

- No interleaving. Systematic coding with interleaving presented in [SPETG18] has no transmission latency, while the decoding latency remains high. Coded packets are transmitted after the entire interleaving block, which is multiple times greater than the generation size. The authors of [AMD20] showed that replacing interleaving techniques with RLC at the link layer can reduce latency, despite losing the ability to recover burst losses.

The adaptive FEC scheme will change either the generation size g, overlap φ, or redundancies per generation r to adjust the code rate Q to the observed loss rate. Given the definition of Q in this work and that each g source packets are protected by $r\varphi$ coded ones, the relationship between coding parameters can be expressed as follows:

$$Q = \frac{g}{r\varphi} \tag{3.1}$$

As previously mentioned, the focus on overlapping generations is intended to uniformly distribute coded packets. This uniformity is achieved by shifting the φ overlapped generations by g/φ source packets, that is, every $g/\varphi = Qr$ source packets a generation is finished and replaced with a new one. Figure 3.1 shows an illustrative example of a communication with 2 overlapped generations of 8 packets, protected with 2 redundancies per generation. The X axis represents source symbols to be transmitted, and the Y axis corresponds to the packets that are actually sent to the network. Source packets are represented with blank squares and cover only one source symbol, while dark squares correspond to coded packets, covering all the source symbols that were used to build it. In this case, it is important to remark that a coded symbol corresponds to one single transmission, regardless of the number of source symbols it covers. Coded packets are inserted every $g/\varphi = 8/2 = 4$ source packets. To keep this periodicity constant from the beginning, the first $\varphi - 1$ generations must be shorter than the intended g. More specifically, $g_i = i \cdot g/\varphi$ if the generations are numbered starting with 1. To comply with overlap definition given earlier, the last overlapping generations also need to be shortened. In the example illustrated in Figure 3.1 this is seen in the last 2 coded packets, which might be unnecessary in a real implementation.

When a lost packet cannot be recovered with a block code, the packets from the corresponding generation can be delivered to the application and discarded. However, when multiple generations are overlapped, all losses can be recovered at once by the

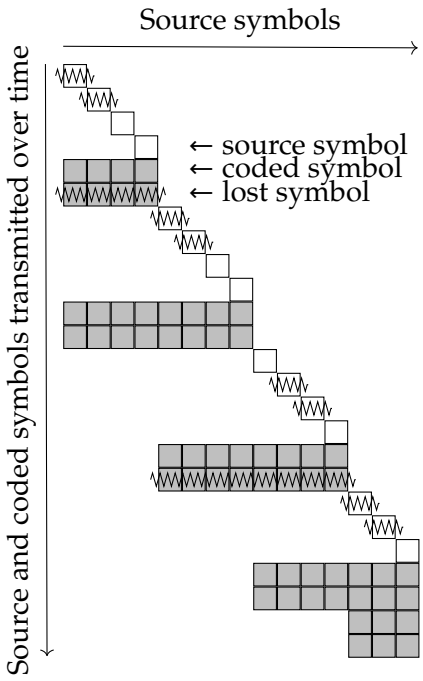


Figure 3.1: Graphical representation of convolutional coding for 15 source symbols protected with 2 overlapping generations of 8 symbols and 2 coded symbols per generation.

communication endpoint. Figure 3.1 shows a short transmission of only 15 source packets with losses. Lost packets are marked with a zig-zag strike-through. Until the last coded packet arrives, no lost source packets can be recovered. This example can be extended to longer communications with different coding parameters. This shows the robustness offered by convolutional coding, which could be useless when FEC is combined with ARQ, since older losses will surely have been recovered with a retransmission. It is thus very important to carefully define coding parameters and the way they are updated in an adaptive coding scheme, because what is beneficial for FEC alone, might be useless when FEC is combined with ARQ, resulting in the transmission of unneeded coded packets and an inefficient increase in computational complexity.

3.2 | Overlapping Generations – Convolutional Coding Rethought

The overview of coding techniques presented in Section 3.1 shows that systematic and convolutional ones help reducing recovery latency. For this reason and to further explore the specifics of convolutional codes that uniformly protect source symbols, a simu-

lation campaign is carried out to evaluate the performance of a particular coding scheme that could be used in applications where the latency is critical. More specifically, this section studies a convolutional Systematic Network Coding (SysNC) focusing on the concurring generations rather than on a coding window. What follows is a description of the coding scheme that has been studied, its evaluation and the results.

3.2.1 | Coding Scheme

SysNC scheme is implemented in Python. Redundant symbols are built with RLNC scheme, combining source symbols multiplied by random coefficients from Galois Field $GF\left(2^8\right)$. To understand how the overlap affects coding efficiency, a constant value is used for each experiment.

In Figure 3.1 it is easy to identify groups of consecutive source symbols between the coded ones. Such a group is referred to as *block*, the consecutive source symbols belonging to the same generation between coded symbols. Generation and block sizes will be referred to as g and k, respectively.

A convolutional coding could recover all losses at once by the end of transmission, as discussed in Section 3.1 and depicted in Figure 3.1. Depending on the particular protocol using this solution and its congestion control scheme, most of these symbols will surely be resent by the transmitter before recovery becomes possible. However, keeping all the received symbols in memory would eventually allow recovering the lost ones. In other words, *recovery probability depends on the amount of memory available at the receiver*. To evaluate recovery potential offered by this coding scheme, the decoding window should be as long as possible. Thus, a sufficiently large buffer is considered to store all source and coded symbols.

The recovery process is triggered after receiving a coded symbol. Given that the channel (described in Section 3.2.2) delivers all symbols in order, this recovery policy is equivalent to trying to recover after each received symbol.

3.2.2 | Channel

For the evaluation of this coding scheme, the simulated connection between the transmitter and receiver is considered wireless, error prone and without packet reordering.

In previous works, the use of uniform distribution for symbol loss modelling is quite common, as in[HPFL09]. However, burst errors might characterize real scenarios. This type of erasure distributions can be mimicked with the well-known Gillbert-Elliot

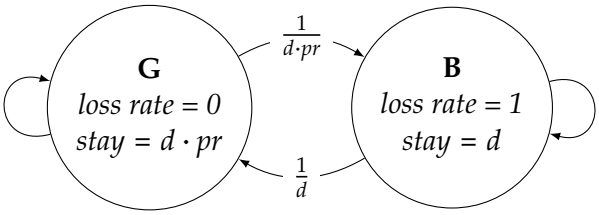


Figure 3.2: Gilbert-Elliot model implementation.

model, as in [WGP+17] and [SPETG18]. In this study both options are considered. Next Gillbert-Elliot model is detailed.

In the Burst state (B) the loss rate is 100%, being 0% in the Gap state (G). Burst state duration is set to d, with the Gap lasting $pr \cdot d$. An overview of this implementation is presented in the Figure 3.2. In this work, pr is adjusted to match an overall loss rate, so the results can be compared with those obtained with a uniform channel model having the same loss rate.

3.2.3 | Inputs and Outputs

Some of the inputs for the communication model used in this work have been already mentioned: generation size, redundant (coded) symbols per generation, overlap, loss rate, burst length, and the number of packets to store at the receiver. At this point one last input needs to be specified: the number of symbols that will be transmitted per experiment. One approach is to keep sending symbols until the decoder can receive (and recover) n source symbols. These experiments follow an alternative approach: n source symbols are sent to the receiver along with the corresponding coded symbols.

In order to see if the coding scheme is working, we need to ascertain whether the receiver has seen all of the source symbols that were transmitted, and if not, how many of them have been correctly received. Thus, the main output is the number of source symbols that have correctly arrived at the receiver, either via regular reception or recovery. Another interesting metric is the ratio between the recovered symbols and the lost source symbols. In this work we evaluate the probability of receiving all (regular reception + recovery) source symbols (henceforth *reception probability*). This probability is calculated by dividing the number of events in which all source symbols have been either received or recovered, by the number of simulations executed for a specific set of inputs, as shown in equation 3.2.

$$P(reception) = \frac{\#\ successfull\ experiments}{\#\ simulations} \tag{3.2}$$

Delay is going to be estimated in terms of time slots required for a packet to reach the receiver. For a correct and reliable data transmission, in-order delivery is highly important. For this reason we focus on the end-to-end delay. In addition, we ignore delay of packets that could not be recovered, and so we only consider for delay estimation those cases where reception probability equals 1.

As stated earlier in this section, our aim is to evaluate the impact of changing redundant symbols per generation with the overlapping SysNC coding scheme. Thus, the two main variables in our input parameters are: redundancies per generation and overlap. In order to facilitate the discussion of results, both variables are combined into one, the *overhead*, which we define as the ratio between the total number of coded symbols and the total number of symbols generated by the transmitter. As can be observed in Figure 3.1, by the end of the transmission $(\varphi - 1) \cdot r$ additional coded symbols are sent. The overall number of coded packets C that protect S source symbols can be calculated as follows:

$$C = \left(\left\lceil \frac{S}{k} \right\rceil + \varphi - 1 \right) \cdot r = \left(\left\lceil \frac{S}{\lfloor g/\varphi \rfloor} \right\rceil + \varphi - 1 \right) \cdot r \tag{3.3}$$

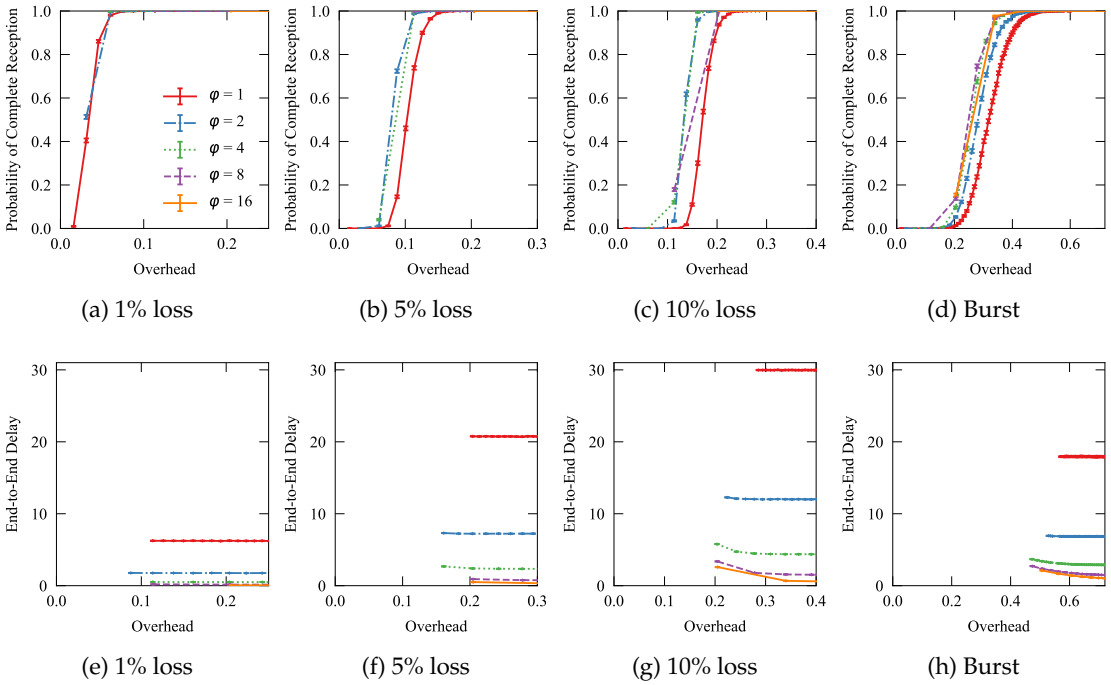
From C, the overhead $\hat{O}$ can be obtained as follows:

$$\hat{O} = \frac{C}{C+S} = \frac{\left(\left\lceil \frac{S}{\lfloor g/\varphi \rfloor} \right\rceil + \varphi - 1 \right) \cdot r}{\left(\left\lceil \frac{S}{\lfloor g/\varphi \rfloor} \right\rceil + \varphi - 1 \right) \cdot r + S} \tag{3.4}$$

Combining coding techniques with a transport layer protocol that uses ARQ makes sense only with relatively small generations. If generations are too big, losses will be recovered with retransmissions requested by a feedback signal. Despite this fact, to better assess the behaviour of this coding scheme when the overlap grows, especially the overhead variations, the use of big generation sizes is more appropriate.

3.2.4 | Results

In order to better understand the impact of both the redundancy per generation and the overlap over the coding scheme, a simulation campaign has been carried out with the following input parameters: generations of $g = 64, 256$ symbols; overlaps of $\varphi = 1, 2, 4, 8, 16$ generations; 2000 source packets (symbols) are transmitted; loss rates of 1%, 5% and 10% with a uniform erasure distribution, and an overall loss rate of 10% with burst losses, having a mean burst duration of 5 time slots (mean gap duration: 45 time slots). In order to ease coding scheme analysis, the memory at the receiver is considered big enough to store all source and coded packets sent by the transmitter. Furthermore,



(a) 1% loss (b) 5% loss (c) 10% loss (d) Burst

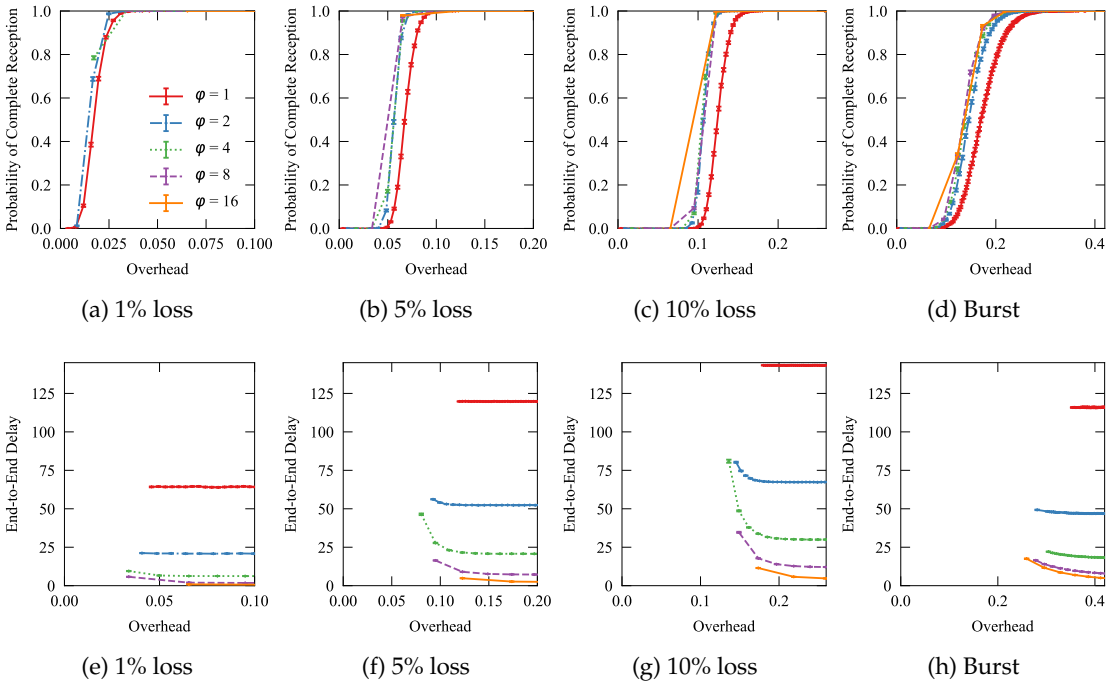
(e) 1% loss (f) 5% loss (g) 10% loss (h) Burst

Figure 3.3: Probability of receiving and recovering transmitted packets (a–d) and end-to-end delays (e–h) for generation size of 64 symbols. The plots are depicted for uniform losses of 1% (a, e), 5% (b, f) and 10% (c, g), and burst losses (d, h).

to ensure statistical tightness of the results, 10000 independent experiments have been run for every configuration.

The results are shown in Figure 3.3 and Figure 3.4. The top rows show the probability of receiving or recovering all source packets for different loss models. The bottom row depicts the mean end-to-end delay. As mentioned in Section 3.2.3, delays are only studied when it is worthy (i.e. all packets were received). We include in all cases the 95% confidence interval.

In Figure 3.3c it can be observed that the reception probability with an overlap of 8 generations is not always higher than the one observed for overlaps of 2 and 4 generations. The same can be seen in Figure 3.3d, where the probability with $\varphi = 16$ at some points is lower than with $\varphi = 4$ and $\varphi = 8$. This is due to the distance between the available samples. The points in which lower overlap schemes have higher reception probability are simply not available for schemes with greater overlap. For a fair comparison we would need to focus only on those overhead values for which the samples of all curves under comparison are defined. At those points where the communication overhead generated by coded symbols is comparable, schemes with higher degrees of



(a) 1% loss (b) 5% loss (c) 10% loss (d) Burst

(e) 1% loss (f) 5% loss (g) 10% loss (h) Burst

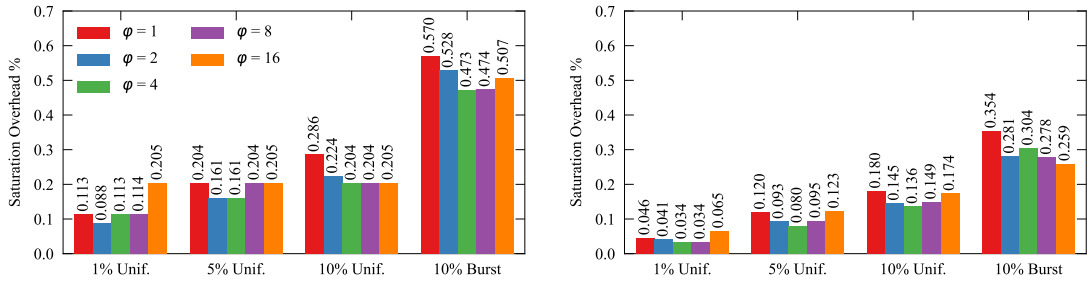
Figure 3.4: Probability of receiving and recovering transmitted packets (a–d) and end-to-end delays (e–h) for generation size of 256 symbols. The plots are depicted for uniform losses of 1% (a, e), 5% (b, f) and 10% (c, g), and burst losses (d, h).

overlap are more likely to receive all the transmitted symbols.

In Figures 3.3 and 3.4 we can see that by having a greater overlap, the reception probability increases and so delay decreases, being the latter the main advantage of SysNC with overlap. According to these results, latency decrease is proportional to φ, as could have been expected, since the blocks (as defined in Section 3.2.1) are precisely φ times smaller than the blocks of SysNC with no overlap. The conclusion of these observations is that within a given overhead, it is more efficient to increase overlap than to have a greater number of redundant symbols per generation.

It is important to keep in mind that *the bigger the overlap, the faster the overhead growth rate becomes, as redundancies per generation increase*. With respect to the possible implementations of this scheme in the IoT field, it should be noted that it might be impossible, due to device limitations or implementation requirements, to increase the overlap. It is thus important to bear in mind that the greater the degree of overlap to be used, the larger the overhead, since there are more redundancies per generation.

An interesting aspect is the overhead at which reception probability reaches its maximum value. We refer to it as *saturation overhead*. This parameter is of special interest



(a) g = 64 (b) g = 256

Figure 3.5: Saturation overhead values for different cases covered by the present study.

when it comes to implement NC in a network with restricted bandwidth. Given that the delay was only studied for those overheads at which reception probability reaches 1, saturation overhead corresponds to the first sample of the delay curve, shown in Figures 3.3 and 3.4. Figure 3.5 shows saturation overhead for all configurations. As can be observed, larger overlap does not necessarily lead to lower saturation overhead. This parameter depends both on the ability of the encoding scheme to recover lost packets, and on the overhead it introduces, so no clear conclusion can be drawn regarding this parameter.

When it comes to choosing overlap and redundancies per generation, it is important to identify the lowest possible saturation overhead. In other words, to find the configuration that will ensure the recovery of most of lost packets with the lowest possible overhead. As the overlap increases, the saturation overhead may increase rather than decrease. By optimizing the saturation overhead, the latency may not be optimal. It is therefore essential to prioritize between channel efficiency (throughput) and delay.

3.2.5 | Summary

This section has studied a convolutional SysNC from an unusual perspective, focusing on overlapping generations rather than on a coding window.

It was observed that full message reception latency reduces proportionally to the applied overlap, which was expected, since this parameter defines the frequency at which coded symbols are inserted. Greater overlap implies a greater overhead. Nevertheless, certain levels of overhead are achieved both by increasing overlap and redundancies per generation. Reaching such values of overhead by increasing overlap is more beneficial for the latency.

In this work we have also identified saturation overhead, which we define as the overhead at which reception probability reaches its maximum. Minimizing saturation overhead optimizes throughput, but not the delay. It is therefore essential to find the trade-off between channel efficiency (throughput) and delay to make the most use of this scheme.

In the absence of feedback about received packets (which reveals the lost ones), this coding scheme is among of the best ones to efficiently recover losses in minimum time. However, extending QUIC with this scheme could be challenging due to the original ARQ loss recovery. When the transmitter knows which are the missing packets, it can retransmit them instead of sending the coded ones, avoiding decoding operations at the receiver and saving time. Next sections detail the design and evaluation of the FEC extension. The integration of coding schemes is further discussed in Section 3.4.

3.3 | rQUIC Architecture

QUIC has been developed considering a traditional client/server communication: the receiver (client) downloads the information sent by the transmitter (server). Also, it is assumed that there may be circumstances (congestion events or packet erasures) where losses occur, and not all transmitted information correctly arrives at the receiver. To recover lost packets, QUIC relies on the classical ARQ mechanism, missing even faster recovery techniques offered by FEC.

This section presents the high-level rQUIC architecture, depicting the interactions between its components. The extension presented in this work applies FEC to short header QUIC packets. Packets with long headers used before handshake is finalized are not modified. Figure 3.6 illustrates the architecture of the proposed extension. The main components are the encoder, integrated within the transmitter operation, and the decoder, which is placed at the receiver's side. The figure also shows the decoder's buffer, which stores packets that might be needed for future decoding operations.

3.3.1 | Encoder

The encoder intercepts short header QUIC packets transmitted by an application, uses them to build coded packets, and inserts the rQUIC header. Furthermore, it uses the congestion window size and losses detected from ACK frames to determine when a coded packet should be transmitted. A more detailed description of the coding operations is provided in Section 3.4.

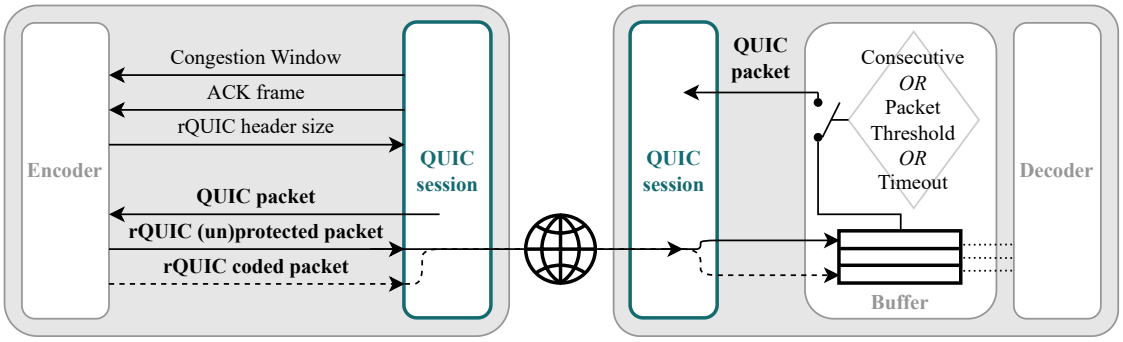


Figure 3.6: rQUIC architecture

The encoder processes packets in different ways. If a QUIC packet carries any frame that would eventually be acknowledged, it will be protected with a coded packet. Otherwise, it does not need to be protected and it will thus be ignored. Hence, rQUIC packet types must be clearly identified. Three main rQUIC packet types are introduced, whose signaling is discussed in more detail in Section 3.5.1.

- Unprotected Packet (UP): they will not be acknowledged by the other endpoint.
- Coded Packet (CP): linear combination of source packets.
- Protected Packet (PP): source packets protected by CPs. These are transmitted, in systematic coding schemes, with minimal FEC signaling, and can thus be used by the application as soon as they are received.

3.3.2 | Decoder

The decoder removes the rQUIC header from the source packets and uses CPs to recover the missing ones. It tries to recover as many packets as possible upon receiving a PP or CP. Depending on the packet type, the following may occur:

- *Unprotected*: the decoder removes the rQUIC header, and forwards the packet to QUIC session.
- *Protected*: the decoder subtracts it from every stored coded packet protecting it. The source packet is then buffered for further decoding operations. Buffer operations are detailed below.
- *Coded*: the decoder checks if it contains all the packets from the corresponding generation. If no packet is missing, the new CP is discarded. Otherwise, the decoder tries to recover any source packet sent by the transmitter, by exploiting the PP it already has. The recovered packets are then stored in the receiver buffer.

3.3.3 | Receiver Buffer

As mentioned previously, the decoder buffers the packets for further decoding operations. In addition, this would also allow FEC module to recover packets that might have been lost. Once source packets are passed to QUIC session, the latter might send an ACK frame[1] reporting all received and missing packet numbers. This could cause packet retransmissions before the decoder can recover recent losses. However, holding source packets for a long time leads to an increase in latency. The decoder's buffer delivers source packets to the corresponding QUIC session when one of the following conditions is met:

- The packet is unprotected.
- The packet is protected, and the previous one has already been delivered to QUIC session.
- The packet is protected, it is the first packet in the buffer, and there are no previous PPs. In this case, the decoder cannot distinguish if there is a missing packet before this PP packet.
- At least 3 PPs[2] from a newer generation were received. At this point it is assumed that generation to which the undelivered packet belongs is already finished.
- The packet has been buffered for too long, and it should be released after a Buffer Timeout (BTO) to avoid compromising QUIC recovery.

Short BTO would make the buffer deliver non-consecutive packets to QUIC session too early, triggering packet retransmission before the decoder was able to recover it. On the other hand, there must be a tradeoff, and BTO should not be too long, as it might delay retransmission when the decoder is not able to recover a loss.

One of the transport parameters defined during connection establishment is the maximum ACK delay [IT20], in which an ACK must be sent for any packet. Therefore, this work defines BTO as the maximum ACK delay D with a certain margin M:

$$BTO := D - M \tag{3.5}$$

The margin M is set to 1 ms. The default maximum ACK delay is 25 ms [IT20]. In the `quic-go` implementation, the endpoints set their ACK delays to 26 ms, and thus BTO in the following experiments is set to 25 ms.

[1] The normal operation establishes that an ACK is sent every two received packets, but this might change depending on the implementation.

[2] 3 is the recommended value for `kPacketThreshold`, packet reordering threshold [IS21].

3.4 | FEC Operations

This section provides details on how FEC operations are performed at both the encoder and decoder, their configuration, and other operations on which they depend.

3.4.1 | Coding Configuration

An encoder builds CPs and decides when to send them. CP building techniques are coding schemes that define how source symbols should be combined to build the coded ones. The proposed rQUIC encoder is designed such that new coding schemes are easy to incorporate. In the implementation used in the experiments reported in this work, both XOR and systematic RLC are supported.

The distribution and frequency of CPs depend on the following parameters, introduced in Section 3.1:

- *Code Rate (Q)*: the rate between source and coded packets.
- *Generation size (g)*: the number of PPs used to create the CP that protects them.
- *Redundancies per generation (r)*: the number of CPs created to protect a generation.
- *Overlap (φ)*: the number of different generations to which any PP belongs at the same time.

rQUIC encoder can adjust the number of CPs it sends based on network conditions (more details are given in Section 3.4.2). To do so, it estimates the code rate Q required to compensate a certain loss rate, estimated from received ACK frames. Then, the generation size g is calculated using equation 3.1. A generation is considered complete when adding a new source packet will make encoder's Q exceeds the required value, or because the generation has reached its maximum size:

$$gen.complete := \left(g + 1 > Qr\varphi\right)\ OR\ \left(g \geq g_{max}\right) \tag{3.6}$$

Redundancies per generation (r) and overlap (φ) are static parameters that are set at the beginning of the connection. As explained in Section 3.1, r CPs are inserted every $g/\varphi = Qr$ PPs. The lower the distance between the CPs, the more often the decoder performs recovery operations. To minimize this distance, r should be set to 1 redundancy per generation.

Big values of overlap φ yield a greater robustness, as well as a generation size increase. Although losses at the beginning of a generation could be recovered by FEC,

ARQ might recover them sooner. Moreover, a greater φ requires more memory at the decoder. For these reasons, the use of a large φ is impractical in protocols with ARQ. The value that minimizes the decoder buffer's length and increases the usefulness of CPs is $\varphi = 1$. However, if QUIC feedback mechanism is changed to optimize the use of FEC, such as replacing ARQ scheme with Hybrid Automatic Repeat reQuest/Query (HARQ), then greater values of φ could improve FEC efficiency. FEC extension described here is meant to extend but not modify existing QUIC functions.

The most practical coding scheme for adaptive code rate, complying with the condition $r = \varphi = 1$ seems to be the traditional XOR used in this study. Other values of r and φ might lead to more efficient configurations, yet more complex interactions with the corresponding congestion control schemes. Research on other configurations is left for future studies.

Upon generation completion, the CPs are sent with the last PP it protects. There is a risk of comprising the transmitter sending rate, known as pacing. [TT21] recommends implementing a pacing mechanism, and `quic-go` has one. However, the impact on pacing should be minimal, when using the XOR scheme with only one CP per generation.

At the receiver side, the decoder will try to recover as many lost packets as possible with every new PP or CP. When a new PP is either received or recovered, it is subtracted from all the received CPs protecting it. Upon each CP reception, the decoder checks if any of corresponding PPs are missing. If no PP is missing, the new CP is discarded. Otherwise, all received PPs corresponding to the new CP are subtracted from it. If the generation of CP suffers more than one loss, the subtraction will not recover them, and the resulting CP will still protect at least two packets. In this case, the decoder will try to decode as many source packets as possible, solving the corresponding system of equations, built with all CPs previously buffered.

3.4.2 | Dynamic code rate

To correct packet losses in environments with unpredictable and varying loss rates, the extension proposed includes the dynamic code rate. The adaptive approach helps recover losses without overloading the link with the CPs. The algorithm used is proposed in [GSF+19, FKCA18].

The code rate Q changes depending on the Residual Loss (RL), defined as the rate between lost and delivered packets. rQUIC encoder takes this information from the received ACK frames. RLs are observed throughout a time period T over N periods, and then the average value is used. As in the original experiment, the period depends on the RTT, $T = 3 \cdot RTT$, and so the number of periods is $N = 3$.

The corresponding dynamic code rate update is presented in Algorithm 1. If the FEC scheme does not prevent losses, Q should be decreased to introduce more CPs. If few (or no) losses are observed, Q should be increased to reduce the redundancies sent to the network. Hence, if the average RL is greater than a certain *threshold* γ, Q is multiplied by $1 - \delta$; otherwise, Q is multiplied by $1 + \delta$, where δ is an arbitrary parameter. In the current implementation, $\gamma = 0.01$ and $\delta = 0.33$.

Algorithm 1 Adaptive code rate in rQUIC

1: $Q \leftarrow Q_{\text{init}}$
2: **if** $RL > \gamma$ **then**
3: $\quad Q \leftarrow Q \times (1 - \delta)$
4: **else**
5: $\quad Q \leftarrow Q \times (1 + \delta)$
6: **end if**

In the absence of losses, Q may grow beyond CWND. If the latter closes, the CP protecting the last PPs will be sent after CWND opens again. If one of the PPs from the last generation is lost before CWND closes, it will not be recovered with FEC. To minimize such cases, the code rate is kept below the CWND (expressed in terms of packets). To convert the CWND from bytes to packets, the maximum QUIC packet size allowed by the implementation is used.

3.5 | rQUIC Implementation Details

Hereafter, the QUIC specification is corresponding to the draft version 29 [IT20], which was implemented in the base code chosen at the time of experimentation.

3.5.1 | Packet Fields

Protecting QUIC packets requires distinguishing coded packets from source packets. In addition, for correct recovery and in-order delivery, the source packets must be appropriately signaled. One approach is to insert these new fields as new frame types. In this way, all rQUIC signaling and coded payloads are encrypted within a QUIC packet. This implies that coding coefficients, a new field, will also be encrypted. However, intra-session NC might use middleboxes (recoders) to improve performance, as discussed in Section 2.2. Although extending QUIC with NC is beyond the scope of this study, the coding coefficients are left unencrypted. This implies that FEC coding is called after

QUIC finishes its own encryption, whereas decoding is performed before QUIC decrypts.

There are other ways to enable the use of NC with QUIC, such as tunneling and the establishment of dedicated QUIC connections for each hop. These solutions are more complex, and the comparison in terms of complexity, connection establishment latency, and power consumption impact on endpoints, such as cybersecure IoT edge devices, define another line of research on their own.

Figure 3.7 illustrates the new fields introduced by rQUIC, which are explained in more detail below.

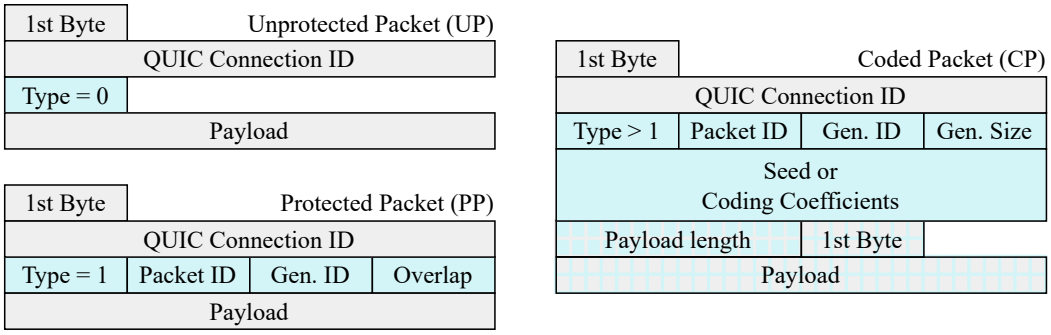


Figure 3.7: New fields for different types of rQUIC packets.

3.5.2 | Headers

The fixed part of the short header packets ends with the Destination Connection ID field [Tho21], after which rQUIC headers are inserted. For correct header insertion, the original QUIC packet size is limited during construction.

The decoder applies different operations to different packet types. Therefore, the first field is the type: PP, UP, and CP. CPs also need to specify the coding scheme used for their creation, that is, coded packet sub-types. In this work, it is used one byte for the type field, including the coding schemes used by the CPs.

All rQUIC packets, in addition to the unprotected ones, need to be identified for correct recovery operations. The next field corresponds to the packet ID, which is 1 byte-length. Packet ID is a sequence number incremented by one after each PP, allowing the identification of PP within the generation. rQUIC reuses old packet IDs. CPs cover multiple PPs and take their IDs from the last PP they protect.

Long generations with a systematic coding scheme might generate CPs that, due to packet ID reuse, might look obsolete. This issue can be overcome by identifying generations using their own IDs. Generation IDs also help the decoder to identify obsolete

packets. This field is also 1 byte long and reuses its IDs. As mentioned in Section 3.1, PPs can belong to multiple generations; therefore, they indicate the most recent generation to which they belong.

Owing to the packet ID field, it is known that the most recent PP a CP is covering. The next CP field after generation ID, which is also 1 byte long, indicates the generation size. rQUIC has been designed to accommodate non-sparse coding schemes. A sparse coding scheme should set this field as if it was covering all PPs between the oldest and the most recent ones, indicating skipped packets with coding coefficients set to 0 in the next field.

Depending on the coding scheme, the CP might need to indicate the coding coefficients that were applied to its PPs, or the seed used to generate them. The size and use of this field depend on both the coding scheme and the generation size, which were included in previous fields. An RLC scheme will use this field to include its coefficients, so its length equals the generation size. On the other hand, the XOR scheme does not use this field because its coding coefficients are always one.

The use of variable-size generations implies dealing with a different number of coding coefficients. For coding schemes that write their coefficients in the protocol header, such as RLC, the corresponding field length would also be dynamic. The longer the generation, the smaller the packet payload. It is not possible to know its size in advance, especially during PP assembly. Based on the results obtained in [GSF+19] (cf. Figure 2 in that paper), the maximum generation size is set to 63, only reached in the absence of losses.

Because the CP header is longer than the PP header, the PP payload should be limited to the maximum size of the CP payload. The implementation proposed in this work, sets the PP header as long as the CP header without coding coefficients, writing in the fourth field the overlap value used by the encoder.

3.5.3 | Payloads

QUIC leaves unprotected the Connection ID field, as well as some bits of the first byte [IT20]. On the other hand, the payload of both the PP and UP goes after the rQUIC header. Because the first byte is partially protected, the FEC must also protect it.

For the CP payload, PP payloads of different lengths can be padded before coding. When transferring large volumes of information or downloading web page objects, most of the packets would be of full length, and only the last one would need padding. However, in the case of an IoT traffic aggregator, with highly variable packet lengths, padding can be much more relevant. An IoT traffic aggregator bundles information

from edge devices into a single flow and sends packets either because they are of full length or upon a timeout expiration. Instead of padding, rQUIC includes the packet length in a dedicated field of 2 bytes.

The coded payload of the CP consists of three fields: (1) the length of the remaining payload, (2) the first byte, and (3) the rest of the QUIC packet after the connection ID. This is the payload that the FEC uses to build a CP, highlighted with a square pattern in Figure 3.7.

To guarantee privacy, endpoints may change their connection IDs, which is the only field that remains unprotected by FEC. If such a change occurs before a generation is complete, the new CP will have a new connection ID. Then, if this is used to recover a PP that was lost before the ID change, the recovered QUIC packet will differ from the original packet in its connection ID. QUIC implementation at the client's side might keep track of the first packet number corresponding to the current connection ID, interpreting older packet numbers sent with the new connection ID as an attack, as QUIC never repeats packet IDs. FEC activity does not aim to undermine the original QUIC operation, nor should it appear to be doing it. Therefore, the encoder checks the destination connection ID, and if it changes, sends all coded packets even if it temporarily increases the coding rate. In this way, every CP will cover PPs with the same connection ID.

3.5.4 | Obsolete Packets

The decoder must keep all packets from a generation to decode its CPs. If generations are overlapped, old CPs combined with recent ones could also recover more recent losses along with the older ones, if the decoder keeps enough buffered packets. Because there might be memory limitations, obsolete packets should be detected and discarded.

As explained in Section 3.5.2, packet and generation IDs, defined in PPs and CPs, are 1 byte long, and their values are reused once the maximum value is reached. The difference between recently arrived packet IDs is expected to be very small; therefore, the comparisons are rather simple. Nevertheless, it is important to define which IDs are considered more recent or older, compared to a specific one. It is considered half of the 1-byte number space for recent IDs, and the other half, for the reference ID and the older ones. As illustrated in Figure 3.8, an ID m is newer than ID n if it lies in the range of consecutive IDs, starting with $n + 1$. The rQUIC implementation uses equation 3.7 to see whether an ID k is older than ID n.

$$(n - k) \,\%\, 256 < 128 \tag{3.7}$$

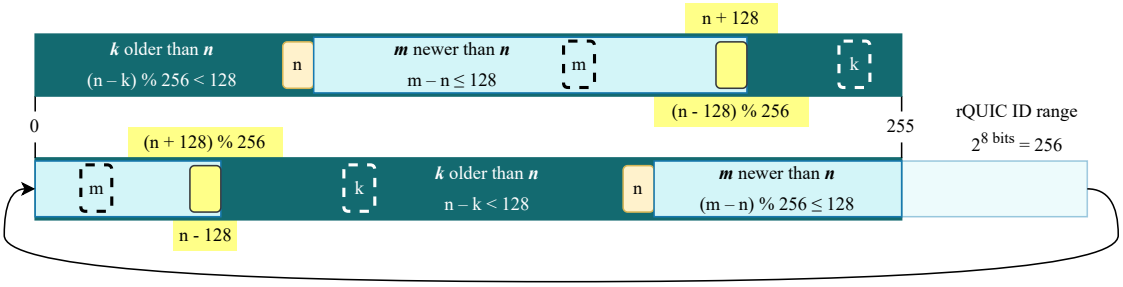


Figure 3.8: rQUIC ID comparison

As was mentioned above, if generations are overlapped, the decoder might not be able to recover lost packets during a long time, until it finally receives enough CPs to recover all losses at once. In a practical implementation, the decoder shall give up on recovering old losses to avoid increasing the latency. A reasonable number of generations is kept, established as the overlap φ plus a margin, set to 1. equation 3.8 is used to obtain the oldest valid generation G, being M the margin of generations to keep.

$$G_{\text{oldest}} = G_{\text{last seen}} + 1 - \varphi - M \tag{3.8}$$

Discarding 'obsolete' packets only based on the generation ID could still leave in the decoder buffer many packets, most of which would not be needed to recover lost and not retransmitted ones. Hence, the decoder will discard all packet IDs older than the oldest allowed one. The decoder calculates the Oldest Valid Packet ID (OVPI) in different moments. First, right after receiving a PP or a CP, the decoder will take its packet ID as the last seen packet ID, and calculate OVPI as shown in equation 3.9. The OVPI is also updated when the decoder detects a packet from an obsolete generation. The new OVPI is the next packet ID after the obsolete one. In both cases the decoder will update the OVPI only if the current value is older, according to equation 3.7.

$$p_{\text{oldest}} = p_{\text{last seen}} - 128 + 1 \tag{3.9}$$

3.5.5 | Assumptions and Design Simplifications

The coding extension for QUIC proposed in this work has been designed to support other schemes, and thus exploit the potential benefits of combining QUIC with, for instance, NC. For this purpose, FEC headers and coding coefficients are added after QUIC encrypts its packets. On the other hand, unencrypted coding coefficients can be used for pollution attacks. Although there are ways to protect the protocol operation against them [ATP20], they can be completely avoided by applying coding techniques before

encryption, leaving coefficients encrypted, as in [MDB19]. Furthermore, unencrypted and recyclable packet IDs can also be used to hinder communication. Coding before encryption not only protects these IDs, but also simplifies the whole process because QUIC packet IDs can be used for FEC.

Even if FEC is applied before encryption, the difference between protected and coded packet lengths may reveal that the communication is protected by the FEC. It also allows the classification of protected and coded packets. The implementation proposed uses shorter headers for PPs than for CPs. By making PP headers as long as CP headers, most PPs will be as long as CPs.

As discussed in Section 3.5.3, there can be cases of frequent short packets not reaching full length. In these situations, even with the same header length for PPs and CPs, a CP will be detectable because it would be as long as the longest PP it is protecting. One possible solution would be to artificially increase the payload of some randomly selected PPs after the encoder has processed them, so that PPs sent to the network seem larger than CPs. Another option would be to randomly split the CP payload into two packets. Their lengths would not be necessarily similar, and if one of them was shorter than the average PP, they could be expanded with random values that would be discarded at the decoder.

One of the key functionalities of CCAs at the transport level is to detect congestion events, and to take appropriate corrective actions, such as limiting the transmission rate. If the encoder is not appropriately configured, it might increase the network congestion with its CPs. Furthermore, it is well known that congestion losses usually occur in bursts, as Cataltepe and Moghe concluded in [CM03]. On the other hand, [KLMW22] suggests that FEC coding could likely benefit communications with persistent non-congestion losses, which is often the case for wireless networks.

Hence, the main assumption of this work is that isolated losses are mostly caused by transmission errors rather than actual congestion. This innovative design also aims to avoid tampering with the operation of CCAs. In this sense, rQUIC is configured to send only one CP per generation, and the generation sizes are limited to the CWND.

3.6 | rQUIC Evaluation Results

This section discussed the evaluation that it was carried out to assess the performance yielded by rQUIC and compare its behavior with that exhibited by the traditional QUIC operation.

3.6.1 | Setup

To carry out the experiments, we exploited the ns-3 simulator[3] version 3.30.1. All networks have the same topology, as depicted in Figure 3.9. The corresponding binary files of rQUIC test applications are placed in lxc containers[4] (Ubuntu Trusty Tahr images), which are connected to ns-3 ghost nodes through CSMA links with large bandwidth and low delay. These ghost nodes are then connected with a point-to-point link, whose characteristics are modified in the experiments to emulate different technologies and network conditions.

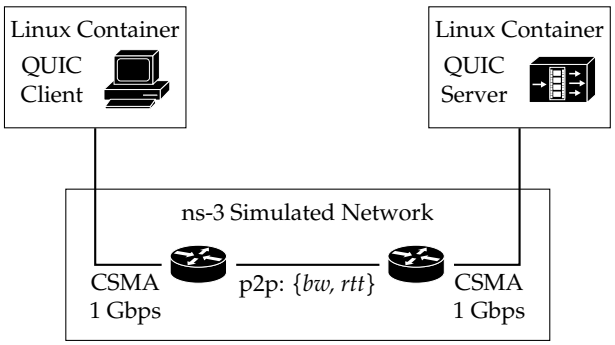


Figure 3.9: Emulation Scenario

The point-to-point link behavior depends on three parameters: BW, RTT, and loss rate. We consider three networking technologies, with different BW and RTT, and then evaluate rQUIC performance over the three of them, modifying the loss rate.

In particular, we selected the same parameters as those used in [GSF+19]. We aim to loosely model the following technologies: (1) Wi-Fi (BW = 20 Mbps, RTT = 25 ms); (2) cellular (10 Mbps, 100 ms); and (3) satellite (1.5 Mbps, 400 ms). In all cases, we introduce different link error rates, α: 0, 1, 2, 3 and 5%.

As discussed earlier, rQUIC can use adaptive or static code rates. In all our experiments, we used adaptive code rate, with residual loss threshold $\gamma = 0.01$ and ratio variation parameter $\delta = 0.33$, as mentioned in Section 3.4. The residual loss measurement period is of 3 RTTs, and it is averaged over 3 measurement periods. As explained in Section 3.4.1, we used XOR as the coding scheme in all experiments, so the number of redundancies per generation (r) and overlap (φ) were both set to 1.

[3] `https://www.nsnam.org/`, Accessed: 23rd April, 2023.

[4] `https://linuxcontainers.org/`, Accessed: 23rd April, 2023.

3.6.2 | Bulk Transfer

The server test application is launched in one container and waits for incoming communications. The client test application connects to the server from another container, and as soon as the connection is established, the server transmits 20 MiB of random data for the Wi-Fi and cellular configurations, and 5 MiB for the satellite.

We then compared the performance exhibited by QUIC and rQUIC by measuring the download completion time, which is defined as the time required to complete the transfer. In addition, to facilitate the comparison between the two protocols, we introduce the completion rate, as defined in equation 3.10. This corresponds to the rQUIC completion time divided by the average of the corresponding QUIC measurements. In this sense, a value of ξ lower than 1 implies that rQUIC outperforms QUIC.

$$\xi = \frac{\text{rQUIC Completion Time}}{\overline{\text{QUIC Completion Time}}} \tag{3.10}$$

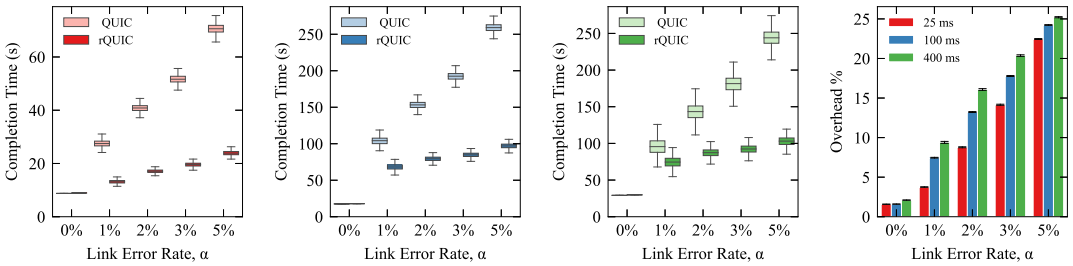
As discussed earlier, a coding scheme (in particular, FEC) aims to improve communication performance at the expense of sending extra packets to the network. Hence, in addition to completion times, we also study the overhead $\hat{O}$ caused by coded packets:

$$\hat{O} = \frac{\text{CP}}{\text{PP} + \text{CP}} \tag{3.11}$$

Figure 3.10 shows the results of the bulk transmission experiment. We ran more than 1150 independent experiments for each configuration. Whisker plots were used to represent the overall delay observed for both QUIC and rQUIC. The boxes represent the interquartile range with the median mark inside of the boxes. The whiskers represent Tukey fences. The overhead bar plots (Figure 3.10d) 95% confidence intervals are represented, although they are difficult to appreciate, since the results are statistically very tight.

For loss rates greater than zero, rQUIC clearly outperforms QUIC. For the Wi-Fi network (Figure 3.10a), the rQUIC completion time is less than half of the QUIC time in most runs. As the loss rate increased, the improvement became more relevant. For instance, when the packet loss rate is 5%, rQUIC completes the 20 MiB downloads in less than 40% of the QUIC completion time. Similar completion time reduction patterns can be observed for the other technologies, where the gain brought by rQUIC becomes more relevant when the loss rate increases.

On the other hand, the figure also shows that rQUIC performs well over ideal channels (0% loss), although the completion time is slightly larger than that shown for QUIC, owing to the small overhead introduced by CPs. Figure 3.10d shows that at a 0% loss



(a) Wi-Fi (20 Mbps, 25 ms) (b) Cellular (10 Mbps, 100 ms) (c) Satellite (1.5 Mbps, 400 ms) (d) Overhead

Figure 3.10: Completion time for bulk transfer and the overhead generated by coded packets.

rate, when there is nothing to recover, rQUIC continues to send some CPs. However, this time increase is very small, and it can hardly be appreciated in Figures 3.10a–3.10c.

Figure 3.10d shows that rQUIC sends more CPs as the loss rate increases. Furthermore, we can also see that more CPs are sent for cellular and satellite technologies, which means that rQUIC observes more losses in these scenarios.

Despite the higher overhead observed for longer RTT values, the results indicate that the gain of the proposed scheme is less relevant for satellite and cellular links, regardless of the loss rate (i.e., when it is greater than 1%). In this sense, the completion time rate stays below 50%, 70%, and 80% for Wi-Fi, cellular, and satellite links, respectively. This behavior can be explained by the decoder's buffer delivering non-consecutive packets (the ones arriving after a loss event) to the QUIC session too early. In these cases, FEC has fewer opportunities to recover lost packets, and thus QUIC reports losses to the transmitter, whose encoder will send CPs more frequently to compensate for the observed losses.

As explained in Section 3.3, the buffer delivers non-consecutive packets to QUIC because either the BTO has been reached, or at least 3 packets from a more recent generation have been received. The latter will contribute to early packet delivery when generations are small, which is a consequence of the increased code rate. The most likely cause of this performance loss is inappropriate BTO values. A more detailed discussion of the impact of BTO on the performance is included in Section 3.6.5.

The above analysis reveals that excessive overhead can impair FEC by making the decoder's buffer see new generations too often. Another consequence of excessive overhead is tampering with the congestion control. To better understand this circumstance, we obtained the average code rate from Figure 3.10d.

The overhead $\hat{O}$ defined in equation 3.11 can be rewritten as follows, where $\bar{g}$ is the average generation size, and $\bar{Q}$ is the average code rate.

$$\hat{O} = \frac{r\varphi}{\bar{g} + r\varphi} = \frac{1}{\frac{\bar{g}}{r\varphi} + 1} = \frac{1}{\bar{Q} + 1} \tag{3.12}$$

From equation 3.12 the average code rate can be calculated as:

$$\bar{Q} = \frac{1}{\hat{O}} - 1 \tag{3.13}$$

Figure 3.10d shows that at a loss rate of 5%, the overhead is above 20%, surpassing 25% for the satellite network. Applying equation 3.13, such overheads correspond to the average code rates of 4 and 3, respectively. Because we are using the XOR scheme, the average generation size is, at a 5% loss rate, below 4 packets, very close to QUIC's recommended minimum CWND of 2 packets [IS21]. Although at this loss rate the rQUIC completion time is less than a half of that achieved by QUIC over all networks, the interaction with congestion control should not be neglected. Hence, the assumption that only 1 redundant packet per generation should prevent worsening congestion events might not be true for network conditions that require high overhead.

Figure 3.11 compares the results with those obtained in [GSF+19] by illustrating the completion rate metric for both cases (the results of our implementation are represented with stronger colors). Confidence intervals have not been represented to simplify the comparison. As can be seen, the rQUIC implementation discussed in this paper exhibits a similar performance to the previous implementation, yielding better behavior when the channel conditions worsen. In addition, the results indicate a more sensible behavior, since the gain with the original QUIC increases for higher loss rates shows a more sensible relationship with the underlying technology (i.e., RTT). In any case, it is worth noting that the QUIC code base was a different one, and that the number of experiments that we used to obtain our results is more than 10 times, which ensures a more precise characterization. In this case, we could not compare the performance of rQUIC with QUIC-FEC [MDB19] because they did not assess the performance of their solution with bulk transfers.

3.6.3 | Webpage Download

This experiment is similar to the one described for bulk transfer. Instead of bulk-data transmission, the client downloads a web page from the server using HTTP. To appropriately emulate web traffic, we used the tool Epload[5], which saves downloaded objects

[5] `http://wprof.cs.washington.edu/spdy/tool/`

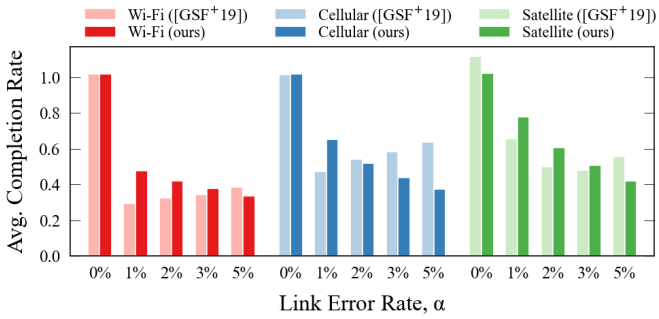
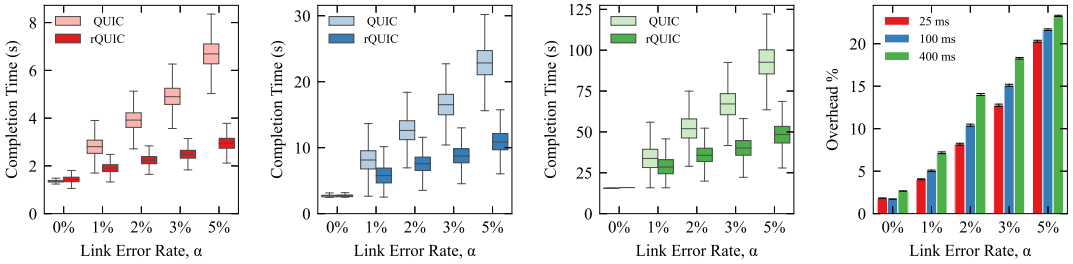


Figure 3.11: Bulk transfer average completion rates for the current rQUIC and its previous version presented in [GSF+19].



(a) Wi-Fi (20 Mbps, 25 ms) (b) Cellular (10 Mbps, 100 ms) (c) Satellite (1.5 Mbps, 400 ms) (d) Overhead

Figure 3.12: Completion time for web page download and the overhead generated by coded packets.

and records a dependency graph. These graphs allow us to precisely mimic webpage download patterns, including the time used to process the objects.

The goal of this experiment was to evaluate rQUIC in short-lived communications with realistic traffic dynamics. We used the flickr.com webpage, obtained from Eproof example data sets[6].

Figure 3.12 shows the results of the web page download experiment. We performed 3500 independent experiments for each configuration, thus ensuring the statistical validity of the results.

The web experiment results show a similar trend to those discussed for bulk transfer. In the presence of losses, rQUIC strongly improves the QUIC completion times. Under ideal conditions, the results were similar.

One aspect that becomes more visible in this experiment (see Figure 3.12a) is that QUIC completion times have much greater dispersion than rQUIC. This aspect might be

[6] `http://wprof.cs.washington.edu/spdy/tool/server.tar.gz`

of interest, since services with stringent real-time requirements should not only ensure a particular average delay, but also an acceptable jitter (variability).

When the QUIC connection is established, the initial CWND grows rapidly until the first loss event. The larger the CWND, the more data is advanced before it shrinks. When the first loss occurs, more or less data will be transferred, having an immediate effect on the connection termination time. This initial CWND variation effect especially impacts short communications, such as the web page download experiment presented here. To better understand this effect, Figure 3.13 illustrates the evolution of the CWND for two QUIC connections over a satellite link, with a 5% loss. The first one (red) reaches large CWND values, and it finishes (completing the web download) in less than 70 seconds, while the second one (blue) suffers a loss at the beginning of the connection, which yields an early CWND shrink, causing the download time to go beyond 110 seconds.

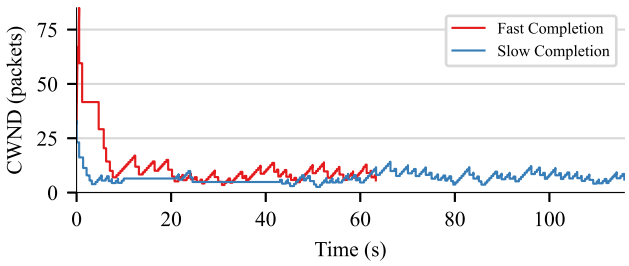


Figure 3.13: Original QUIC CWND evolution for a fast and a slow web page download over the satellite link at 5% loss.

The main reason for dispersion reduction with rQUIC is that FEC recovery prevents CWND from shrinking in the presence of a stable loss rate, and thus delays a strong CWND shrink at the beginning of a communication.

The overhead behavior (both in terms of network technology and loss rate), illustrated in Figure 3.12d is similar to that discussed for the bulk transfer experiment. We can see that the web experiment yields lower overhead. This is due to the shorter communication duration, as well as to the initial generation size, which is as large as the initial CWND, that is, 32 packets. This results in an overhead that might not be sufficient for certain network conditions.

As we also observed in the bulk experiment, Figure 3.12d shows that, as the RTT increases, the overhead increases, although, as can be seen in Figures 3.12a–3.12c, rQUIC completion time does not necessarily improve, compared to QUIC. The reason for this overhead increase, without a clear FEC performance improvement, is likely the same as that discussed for the bulk transfer experiment: a too short BTO. Further discussion on BTO is included in Section 3.6.5.

As was done for the bulk experiment, Fig 3.14 compares the performance exhibited by the rQUIC implementation discussed here with the one discussed in [GSF+19]. In this case, we can see that the corresponding completion rates exhibit much better behavior. Again, the relationship between both the loss rate and RTT is more sensible. In this case, the behavior of the cellular and, especially, the satellite technologies is clearly better than that obtained in [GSF+19]. Again, it is worth highlighting that, although the characteristics of the experiment are similar, we have used a more recent QUIC code base, and the number of experiments is notably higher than those that were run in [GSF+19].

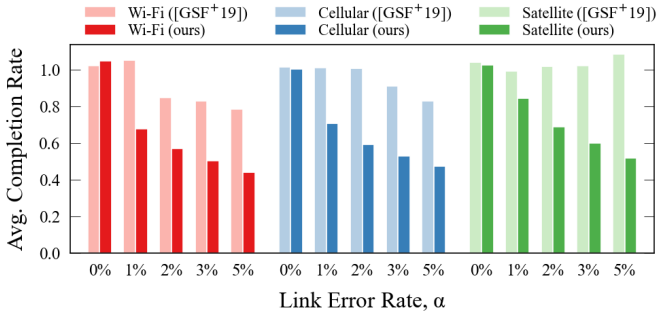
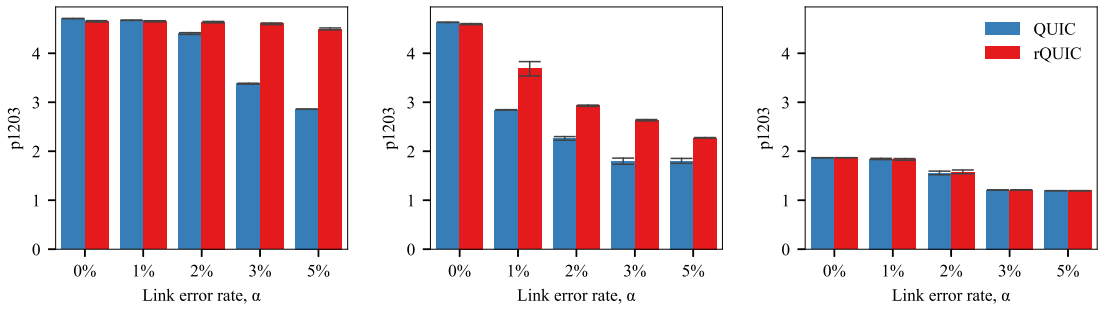


Figure 3.14: Web page download average completion rates for the current rQUIC and its previous version presented in [GSF+19]

The results presented by Michel et al. in QUIC-FEC [MDB19] are not directly comparable with ours owing to different setups and coding schemes. They only used two scenarios: (1) Direct Air-To-Ground Communication, with a bandwidth of 0.468 Mbps, where FEC yielded a worse performance, due to a large overhead of 33.33%. (2) Mobile Satellite Services, with a bandwidth of 1.89 Mbps, an RTT of 761 ms and a loss rate of 6%. In the latter case, the 1 MB file transfer is somewhat similar to our web page download experiment, where less than 2 MB is transferred with a bandwidth of 1.5 Mbps, an RTT of 400 ms, and a maximum loss rate of 5%. The rest of the experiments carried out in [MDB19] imply the transmission of short files, and they are thus not comparable with our results. While QUIC-FEC [MDB19] increases the completion time owing to its relevant overhead, the rQUIC adaption scheme maintains the overhead at a reasonable level, slightly greater than 25%, leading to a significant improvement compared to the original QUIC. In any case, the setup parameters are not the same, and both implementations take different trade-offs: QUIC-FEC sacrifices bandwidth by sending more redundant information, while rQUIC worsens congestion awareness by masking losses from congestion control. Thus, we can conclude that a combination of an adaptive coding scheme, full congestion awareness, and an advanced CCA could bring additional benefits.



(a) Wi-Fi (20 Mbps, 25 ms) (b) Cellular (10 Mbps, 100 ms) (c) Satellite (1.5 Mbps, 400 ms)

Figure 3.15: QUIC and rQUIC performance comparison using p1203 score.

3.6.4 | Video Streaming

The last bunch of experiments uses video streaming to ascertain whether rQUIC can yield some gains with this real traffic.

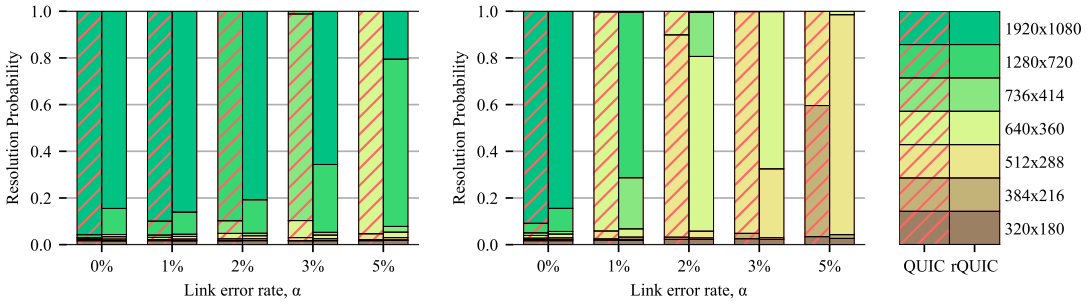
Dynamic Adaptive Streaming over HyperText Transfer Protocol (DASH) is a standardized technique [II19] for sending video streams over HTTP. We used `go-dash`[7], an implementation in the GO language of the DASH protocol, published in [RMQ20], as well as a DASH testbed[8].

The DASH server streams "Tears of Steel" short movie. We used the p1203 [RGR+17] Quality of Experience (QoE) parameter to compare rQUIC with QUIC. To better understand the benefits of rQUIC, we also performed a qualitative comparison, by studying the video resolution that was used under different network conditions.

Figure 3.15 depicts the average p1203 score with 95% confidence intervals for DASH video streaming. For the Wi-Fi network (Figure 3.15a), the results indicate that rQUIC outperforms QUIC when network conditions worsen (link error rate greater than 1%). In the other two cases, the performance is almost similar to that exhibited by QUIC. In the cellular network, rQUIC is able to yield a higher score than QUIC in all cases, but for the ideal channel, where the performance is almost alike (i.e., rQUIC does not hinder the performance exhibited by the original QUIC). These observations are aligned with those observed in the two other experiments (bulk and web traffic). For the satellite link, there is almost no difference between the two protocols, and rQUIC does not yield any improvement compared to QUIC, as can be seen on Figure 3.15c. In any case, the scores are rather low, implying that this technology might not be able to provide an appropriate quality of service for this type of real-time application.

[7] `https://github.com/uccmisl/godash`
[8] `https://github.com/uccmisl/godashbed`

To complement the previous p1203 score results, Figure 3.16 depicts the video resolution probability distribution. The aim is to quantitatively show how rQUIC can outperform QUIC when using video streaming services by allowing the transmission of higher quality frames. Because the results that were observed for the satellite technology evince rather low quality for both rQUIC and QUIC, we only illustrate, in Figure 3.16, the results obtained for Wi-Fi and cellular technologies.



(a) Wi-Fi (20 Mbps, 25 ms) (b) Cellular (10 Mbps, 100 ms)

Figure 3.16: QUIC and rQUIC performance comparison in terms of achieved video resolutions.

Figure 3.16a shows that when the loss rate is ≥ 2%, rQUIC is able to transmit higher resolution frames than QUIC. Even for the worst conditions (5% loss rate), rQUIC can maintain most of the resolutions at 1280×720, while with QUIC most of the time the transmitted resolution is 640×360. For better quality channels (0% and 1% loss rate), the resolutions for both QUIC and rQUIC are rather similar, corresponding to the p1203 score that was discussed earlier.

The same behavior is observed over the cellular link, where rQUIC clearly yields better resolutions than QUIC when the conditions of the underlying links become worse (Figure 3.16b). In this case, we observe this behavior for all values of loss rate, but for the ideal case (loss rate 0%), where the resolutions that were seen for the two transport protocols are almost alike.

3.6.5 | Buffer Timeout Exploration

We observed that the FEC performance worsened for larger RTT values. Our initial assumption is that this degradation is caused by an inappropriate BTO value. However, using a fixed BTO may not be a good solution for changing network conditions.

If packet transmission rate is paced, the optimum BTO can be searched according to the pacing algorithm. Based on the recommendations in [IS21], `quic-go` implements its

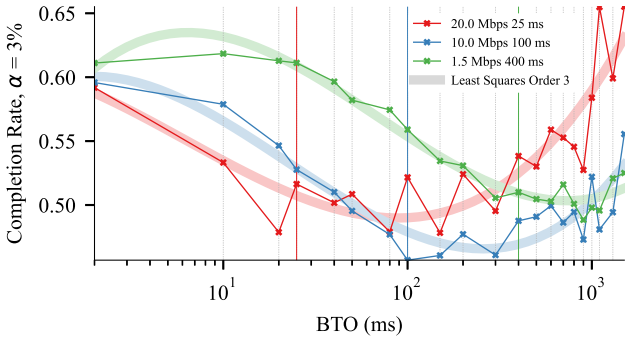


Figure 3.17: Average completion rates for different values of BTO in web page download.

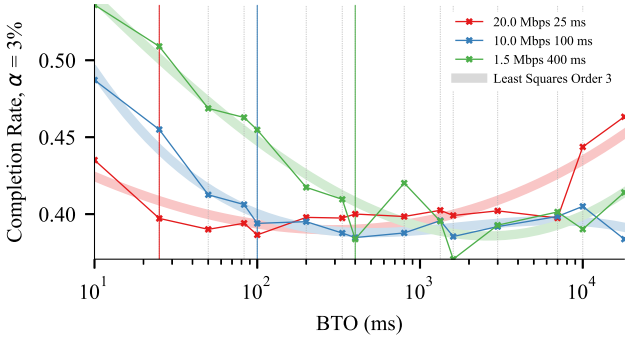


Figure 3.18: Average completion rates for different values of BTO in bulk transfer scenario.

sending rate as shown in equation 3.14, where sRTT corresponds to a smoothed RTT. By default, the endpoints do not communicate their congestion windows, but each endpoint estimates the RTT. Thus, defining the BTO in terms of sRTT could yield a nearly optimum value.

$$sending\ rate = \frac{5}{4}\frac{\mathrm{CWND}}{\mathrm{sRTT}} \tag{3.14}$$

To better understand the impact of different BTO values on the rQUIC performance, we repeated the bulk transfer and web page experiments with different BTO values.

Figures 3.17 and 3.18 show the completion time rate (ξ) for a loss rate of 3%. The figures show, with a vertical line, the RTTs of the three networks. The default BTO is 25 ms, corresponding to the RTT for the Wi-Fi network. Since there was some dispersion in the results, caused by the dependency on both the congestion control algorithm and the network conditions (loss rate), we have included, as thick lines, the 3rd order least squares regression of the results observed for different BTO values.

The corresponding trend would ease the identification of an optimum point, where the BTO would yield a higher gain (shorter completion time).

Since the optimum BTO depends on the sending rate, impacted itself by the RTT, it would be reasonable to expect that a BTO equal or proportional to the RTT would yield a minimum completion rate. Figures 3.17 and 3.18 show the reduction in completion rates for cellular and satellite links when BTO equals the underlying RTT. However, the results indicate that lower completion rates might be obtained with BTOs greater than RTT.

We can see that there is no clear or constant relationship between the BTO that yields the minimum completion time and the corresponding RTT for the web traffic experiments. In this sense, the minimum values in Figure 3.17 are observed at, approximately, 90, 240 and 700 ms for Wi-Fi, cellular and satellite links, respectively, 3.6, 2.4 and 1.75 times greater than their corresponding RTTs. Bulk transfer experiments neither yield a constant BTO to RTT relationship, with minimum completion rates in Figure 3.18 found around 300 (12 RTT), 400 (4 RTT), and 2500 ms (6.25 RTT) for Wi-Fi, cellular, and satellite links, respectively. Thus we can conclude that the optimum BTO cannot be defined as a function of RTT alone. However, a BTO that is equal to RTT seems to be a safe and efficient solution for both short (web) and long (bulk) communications, as the difference with the potential optimum performance is not very relevant.

On the other hand, excessively large BTO values would jeopardize the overall communication delay. Figures 3.17 and 3.18 show that the completion time rates for the three considered technologies grow as the BTO increases. This is due to the highest BTO values used for the test, which are comparable to the completion times in this particular scenario. On the other hand, BTO values are approximately one order of magnitude smaller than the completion times (see Figures 3.10 and 3.12) are greater than the optimum BTO, and completion rates are growing (see Figures 3.18 and 3.17). This growth is more difficult to appreciate in Figure 3.18 for cellular and satellite links, because the communication is longer, and the observed growth rate is still low.

3.7 | Summary

This chapter has focused on integrating FEC functionality in a modern transport layer communication protocol, namely QUIC. First, a thorough evaluation of coding schemes has been done to choose those that would help minimize loss recovery latency. Next, we presented the design and implementation of rQUIC, an integration of QUIC with a FEC module. Our proposal can be configured to consider different coding schemes,

including various Network Coding flavors. The implementation in the go language has been made available in a public repository.

We assessed the performance of our proposed method by comparing it with that exhibited by the original QUIC protocol. We exploited the ns-3 simulation framework, which by means of virtualization, allows the integration of containers hosting real nodes. Thus, we were able to use realistic traffic patterns. The simulator also allowed us to perform repetitive and systematic experiments, in which different technologies and conditions (link qualities) were considered. We used complementary traffic patterns, embracing both long (with bulk data transmission) and short flows (typical for web transfers). In the two cases, rQUIC clearly outperformed the original QUIC protocol, as well as previous works that also integrated QUIC with a coding module. Furthermore, we have also studied the benefits that the coding module could bring for a real-time service, by integrating our proposal with the DASH protocol. In this case, the results show that in scenarios where the video stream quality is reasonable, the use of rQUIC increases the QoE perceived by the end user, allowing the transmission of frames with a higher resolution.

To further enhance QUIC, we add MP functionality. There already is an ongoing effort to define a basic MP for QUIC in the IETF [LMC+23], however few works have evaluated the combination of QUIC, FEC and MP [VW21]. Next chapter details our MP implementation and how MP and FEC can be combined to simultaneously boost reliability and reduce latency.

4

Multipath for Transport Protocols

MP techniques consist in using multiple paths through the network to connect the endpoints. The endpoints are not able to choose the exact network path, *i.e.*, the routers that will be traversed, nevertheless they can simultaneously use multiple network interfaces if more than one are available. The resulting network paths would likely overlap, but not in the close vicinity of the endpoints using multiple network interfaces.

MPTCP defines a path as a sequence of links between a sender and a receiver, identified at transport layer by a 4-tuple of source and destination address/port pairs, and the data transmitted on a path is referred to as subflow [FRH+20]. An endpoint could establish multiple paths over the same network interface using different ports, creating overlapping sequences of links. Thus, it is necessary to distinguish between *network path*, a sequence of links, and *logical path*, SoftWare (SW) defined path with its own unique identification, the 4-tuple in case of MPTCP.

The first attempts of handling MP connections consisted in wrapping different SP connections to present them as only one. TCP derived protocol pTCP follows this approach, wrapping TCP sockets in one socket available to the application [HS02]. However, transport layer connection need more synchronization than a simple wrapper. CC applied separately to each transport connection adapts poorly if an intermediate node used by multiple paths gets congested. The paths that are not the first ones to detect the congestion will drop their packet transmission rates with a delay in which they will be worsening that congestion. Moreover, uncoupled CCAs are not pareto-optimal: on shared MP links the paths will use network resources as multiple SP connections, which could result in throughput reduction of regular SP connections [LLO+16, JB22].

Probably the easiest approach to implement a MP extension is making maximal use of SP code. MP can be seen as 2 or more SP connections, as in pTCP, but with shared (synchronized) management, which includes flow and congestion control. Current draft

specification for QUIC MP extension clearly states that it intends to maximally reuse existing SP QUIC specification [LMC+23].

The architectures of original and extended QUIC are represented in Figure 4.1. The rest of the chapter reviews QUIC architecture identifying the most relevant components and describes technical details of our combined MP and FEC extension. To implement our MP extension we follow both the specification [LMC+23], as well as the public discussion [IIa, IIb]. We focus on identifying connection management, which ensures the original SP connection. We describe how this connection management is reused to create new paths and the way multiple paths are synchronized. Then we describe the integration of FEC in MP environment. For MP experiments we use the same `quic-go` implementation as in Chapter 3.

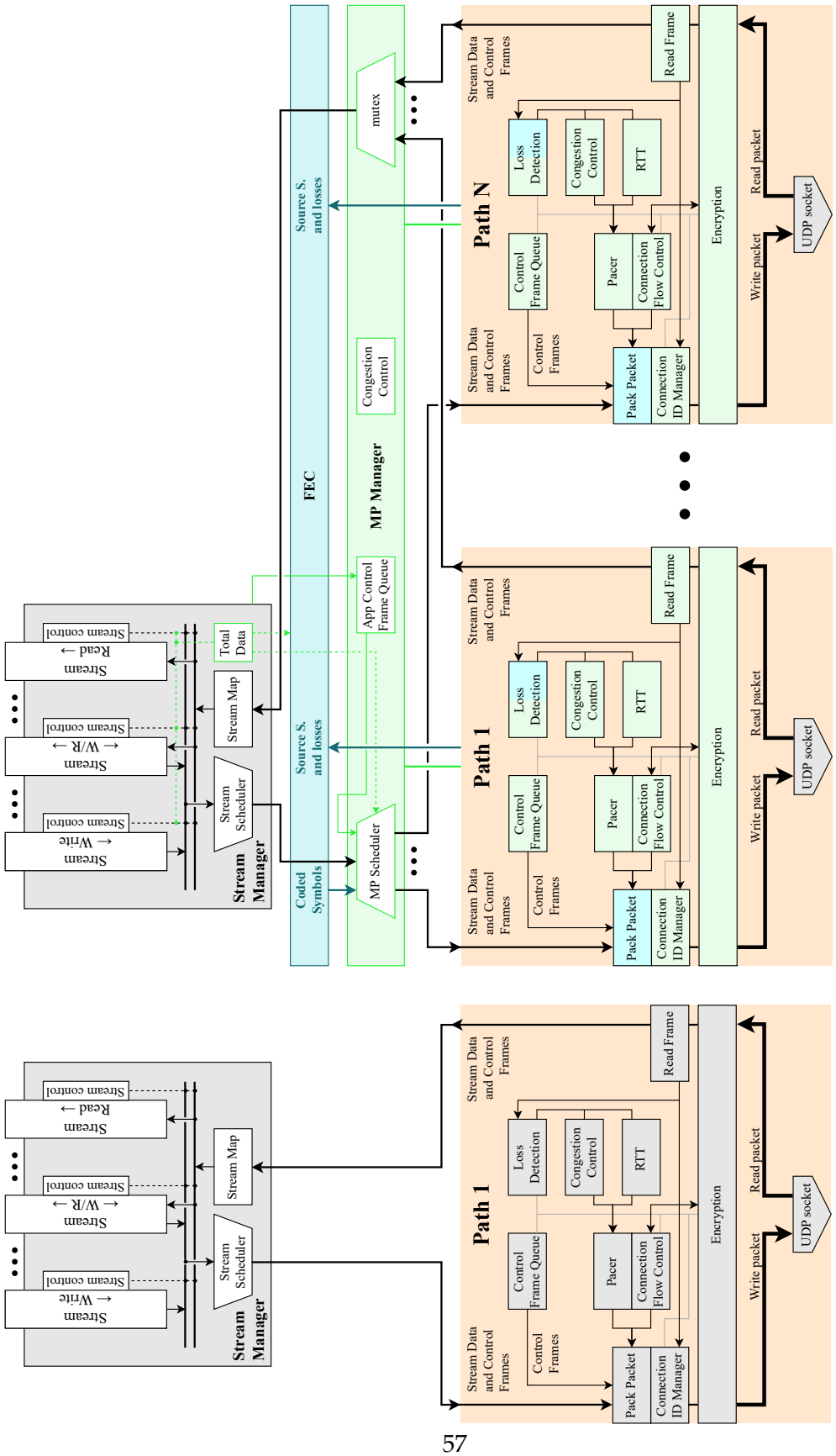
4.1 | QUIC Components

Figure 4.1a shows the most important components of original QUIC that will be affected by FEC and MP extensions. These components form two groups: data and connection management.

QUIC was designed mainly for HTTP/2 traffic, which transfers data from multiple sources in multiplexed streams to limit the HoLB problem [LIB+17]. To keep the streams multiplexed, the retransmission of lost data is managed by the corresponding streams. To further avoid HoLB, QUIC limits the receive buffer not only for the connection, but also for each stream. If an application reads a specific stream's data slowly, per-stream flow control won't let this stream fill the whole receive buffer dedicated to QUIC connection. A QUIC receiver announces the available byte offset for each stream. If the transmitter reaches the offset without receiving an updated one, the transmission on that stream halts [LIB+17, IT21].

In Figure 4.1a we represent the stream manager in the upper part. It is comprised of streams, with flow and retransmission control attached next to them, stream scheduling which selects stream data for the next packet, and the stream map, which delivers stream specific frames to the corresponding stream or its controller. The implementation we use also supports datagram extension [PKS22]. However, we do not represent the datagram queue in Figure 4.1 because we do not consider using them neither to test MP nor FEC.

Connection management comprises all those components that ensure and manage the connection regardless of the streams. Connection management is what we reuse to create new paths. This is why in Figure 4.1a (bottom) we call this group "Path 1".



(a) SP QUIC. (b) MP QUIC with FEC.

Figure 4.1: Architecture of the original SP QUIC and the one extended with MP and FEC.

In the moment QUIC detects it can send a packet, a packet packer prepares an empty packet and fills it with control, and stream control and data frames. `quic-go` implementation always prioritizes control frames over others, and lost frames (retransmissions) over the new ones. Once the packet is filled, it is encrypted. After building certain number of packets QUIC may change its Connection ID (CID) for security reasons. Another reason to change CID is because the endpoints have detected a change in the IP address [IT21, Section 8.2].

QUIC specification recommends using a pacer, the component that calculates next packet transmission time [IS21]. `quic-go` includes a simple pacer that tries to match the estimated BW. The latter is calculated as the ratio between CWND and RTT multiplied by a correction factor of 5/4 to not to underutilize the actual BW. QUIC measures its CWND in bytes, not packets. If the previous packet was short, the time to wait before sending a new one will also be shorter. As soon as the next transmission time is reached, a new packet is built and sent. Pacer is not the only component limiting packet transmissions. If the limits of CC (CWND) and connection flow control are reached, QUIC will not send new data.

Just like TCP, QUIC uses ARQ to confirm received packet and detect the lost ones. A more complex and efficient ACK transmission is not the only difference between the two protocols. While TCP has only one transmission queue, QUIC has one for each sending stream and requires the implementation of another one for non-stream control frames. Upon a packet loss, its frames are enqueued in the corresponding queues.

QUIC control signals are packed as frames [IT21], unlike TCP, which sends them in packets' headers [Edd22]. Control frames can be produced by almost any component we represented in Figure 4.1a. For instance, a change in peer's IP address is detected at packet reception. Whether it changed because of NAT rebinding or an intentional path migration, the new IP triggers path validation carried out with PATH_CHALLENGE and PATH_RESPONSE frames, and a change in CID, which might trigger issuing of new CIDs via a NEW_CONNECTION_ID frame, or at least retire the one in use with the RETIRE_CONNECTION_ID frame [IT21]. In Figure 4.1a we connect the components interacting with control frame queue with a grey line.

4.2 | Parallel Connections

With the idea of reusing as much of SP design as possible, we think of MP as parallel SP connections. The idea of parallelism raises the concern about the kind of devices that will be running such a code. Low-end IoT devices, such as low-power microcon-

trollers, will surely have only one network interface. A device running any MP protocol will have at least two network interfaces and have enough memory and computational power to handle multiple connections. On the othe hand, growing IoT applications are becoming more complex. For instance, a warehouse management system can use multiple unmanned vehicles with powerful processors to navigate safely avoiding collisions. Nowadays smartphones include at least dual-core processors. Thus, we conclude that parallel SP connections can be implemented as actually parallel threads running simultaneously. To open a new path, in the base QUIC implementation we replicate the connection object, as represented in Figure 4.1b.

4.2.1 | Data Streams

The stream manager does not to change to send data to multiple paths. It would be logical to expect that MP scheduler would properly manage the incoming data stream. On the other side, MP scheduler could use some information from the stream manager for better scheduling decisions. As we discuss in Section 4.3, we take the total amount of data bytes to send and retransmit in each sending stream. On the other hand, it has been proved that stream and MP schedulers perform better when aware of one another. Rabitsch *et al.* designed a stream aware MP scheduler [RHB18], reducing page load time on heterogeneous paths. In 2019 Shi *et al.* proposed a stream scheduler that selects a path for each stream, making a SP transmission for each stream [SWZL19]. With a prioritized stream of 26 KB they observed a significant benefit of sending it only on one path, avoiding MP latency due to different completion times on heterogeneous paths. In 2020 Shi *et al.* proposed another stream-MP scheduler, focused on the earliest completion of the prioritized streams that utilizes all paths that would not delay stream completion time [SWZ+20].

4.2.2 | Build Packets

The packet packer component could need significant changes if it would have to build packets for all paths. However, given that each path has its own dedicated connection construct, this component is replicated and works exclusively with the corresponding path, unaware of other paths. Apparently, the only extension of this component is enabling the interaction with FEC to signal source symbols and insert the coded ones.

QUIC control frames are queued in the control frame queue, which is path specific. Stream specific frames are managed by stream controllers with their own queues. However, other application data related control frames, such as MAX_STREAMS and

STREAMS_BLOCKED frames, have to be enqueued in one of the control frame queues. Our proposed solution is to queue such frames in a dedicated queue managed by MP manager. This queue would be available to the path with the lowest RTT. Upon a new packet building, after packing path specific control frames the packet packer would try to get frames from this queue.

MP draft specification does not introduce big changes to the encryption. In the first version of the unified MP extension draft The nonce for packet protection was combined (XOR operation) with path ID [LMC+22]. The latest version of this draft (04 at the time of writing) forces the use of CID sequence number as the path ID [LMC+23]. The draft also specifies how key update needs to be synchronized on all paths. More specifically, if packet protection key update began on one path, at least one packet with the updated keys has to be sent on the other paths. New key updates are not allowed until a packet with current key is acknowledged on every path.

4.2.3 | Path and Connection IDs

To manage CIDs, QUIC refers to them with sequence numbers, assigned by the issuing endpoint. As already mentioned, MP draft specification uses CID sequence numbers as path identifiers. Given that each path is going to use its own CID, the sequence numbers of the latter can perfectly and uniquely identify each path. Linking path identification with CID results in the decision to reset packet number space every time a new CID is used [LMC+23]. A client might change its CID and 4-tuple for privacy reasons [IT21, Section 9.5 Paragraph 8]. The server would see a CID it has issued but which hasn't been used so far, and with the new 4-tuple it will not be able to map the new packets to an existing path. QUIC packet numbers are integers in the range 0 to $2^{62} - 1$, but the corresponding field in packets' headers is limited to 32 bits [IT21, Section 17.1]. For this reason, if a packet cannot be mapped to an existing path, it is impossible to know the previous packet number necessary to infer the full packet number, without which the packet cannot be decrypted. A generic algorithm that would deal with these cases would reset packet numbers each time a new CID is used [IIb, Issue 182[1]].

The original QUIC does not reset packet number space because all CIDs belong to the same SP connection. To preserve this behaviour, CIDs should be assigned to specific paths. NEW_CONNECTION_ID frames can force the receiving endpoint to retire CIDs prior to the one specified with its sequence number. In a SP connection these numbers can be consecutive. In a MP connection that shares all available CIDs among all paths, the sequence numbers will not be that consecutive. Using an explicit path identifier,

[1] `https://github.com/quicwg/multipath/issues/182`, Accessed: 23rd April, 2023.

among other improvements, would greatly simplify CID management, remove the need to reset packet number spaces and enable unambiguous path migration identical to SP QUIC [IIb, Issue 214[2]]. With a separate path ID there would be a separate CID sequence number space for each path.

To completely isolate CIDs for each path, path-opening endpoint should provide the other one with at least one fresh CID to be used on a new path. We propose a simpler solution: to open a new path, the endpoint takes the unused CID with the smallest sequence number on the main path, which would keep an ordered consumption of CIDs on the main path. The newly opened path would start with the sequence number of the CID taken from the main path, and the other endpoint would issue CIDs starting with the next sequence number. On one hand the path ID could be the first sequence number used on the path. On the other hand, if a path starts late in a long lived connection, the new sequence number defined as a variable integer could be longer than one byte. We do not foresee the use of 64 concurrent paths in the nearest future, which is the maximum range offered by 1-byte variable integer. We define path ID as a variable integer, but do not link it to the first CID sequence number. Each newly open path will increase the last ID by one. The first ID corresponding to the main path is zero. Trying to open a path with an ID that has been used should generate a dedicated transport error.

4.2.4 | Transmission Rate Control

As explained in Section 4.1, the pacer limits transmission rate based on CWND and RTT. To keep the right transmission rate, there should be a separate pacer for each path. Nevertheless we need to introduce one modification in this component: it has to report the average time between packets (or the average throughput).

The CCA we use is OLIA [KGPL13], reusing its implementation from [DCB17]. Each path uses its own instance of the CCA, all of which are linked through a master object. In [JB22] researchers evaluated the performance of MPTCP using coupled CCAs and compared it with SP TCP using New Reno CCA. They concluded that CCAs Linked Increase Algorithm (LIA), Opportunistic Linked Increase Algorithm (OLIA) and BAlanced LInked Adaptation (BALIA) were underutilizing the available BW when a congestion event only affects one path. Despite the need for further research in CCAs for MP, this effort is beyond the scope of the present work.

The goal of QUIC connection flow control is to prevent fast senders from overwhelming the receive buffer [IT21, Section 4]. Given that QUIC connections have more than one data sources (namely streams), QUIC also includes stream flow control, which

[2] `https://github.com/quicwg/multipath/issues/214`, Accessed: 23rd April, 2023.

prevents a single stream from occupying the whole receive buffer. Extending QUIC connection to multiple paths introduces the need for another level of flow control. The receiving endpoint should be able to allocate more memory for each path, at least as much as for the main path, and ideally, as much as twice the product of aggregate BW and the maximum RTT [IRB+11, Section 5.3]. Separate connection flow control for each path should work as in SP QUIC. However, a slow path would be underutilizing its portion of buffer compared to a faster path. MP flow control could distribute connection (path) receive windows based on the average packet reception rate. MP HoLB prevention can be achieved by FEC [LLT+14] or a proper MP scheduler that reorders data during transmission to optimize in-order reception [SBL+13, KLM+14, SCW+18]. If HoLB prevention is already implemented and sufficient memory is allocated for the receive buffer, MP level memory optimization might not achieve significant improvements, nevertheless giving more margin for HoLB prevention.

4.2.5 | RTT and Loss Detection

[LMC+23] recommends sending the ACK on the same path where the acknowledged packets were received. We refer to this strategy as *default* ACK strategy. Its use results in path RTT measurements. Sending an ACK on a different path would measure what we could call a cross-path RTT, which combines one-way delays of different paths. The availability of multiple paths for sending ACKs can complicate RTT estimation, as discussed in [Hui21b, Section 3.3]. A MP implementation should support sending ACKs on different paths in case one of them becomes unavailable in the ACK direction. Moreover, full support of cross-path RTTs can be beneficial for MP packet scheduling.

The knowledge of one-way delays could improve MP scheduling. Estimating it as half of the path RTT is sensible, although not precise, given that network routing in each direction could differ. As discussed in [Hui21a], measuring all path and cross-path RTTs will neither provide one-way delay estimations. The only way to measure them is using a timestamp extension [Hui22], and these measurements are not free of the endpoints' clock synchronization errors. On the other hand, more than the actual one-way delay, MP scheduler needs the difference between these delays. Sending all ACKs on the same path will result in each path's working RTT estimation (the RTT components in Figure 4.1b) being defined with its own transmission one-way delay and the ACK path delay, which is common for all paths. We refer to this strategy as *one-path* ACK strategy. With its application, the difference between the working RTTs is the same as the difference between the transmission one-way delays. We believe that to keep the RTT estimation consistent a receiver should use either the default or one-path strategy.

A MP implementation should distinguish between all path and cross-path RTTs. Otherwise, an ACK sent on a different path than usually will worsen RTT estimation precision and increase its variance. Nevertheless, tracking all of possible RTTs can be used to improve the connection. Using one-path ACK strategy, the receiving endpoint could ping-probe each path to determine the fastest one for acknowledging the incoming traffic. If for any reason a path becomes unavailable in the ACK direction, the ACKs can be easily sent on another path. If at this moment all path and cross-path RTTs have been sampled, the RTT update will consist in relying on the corresponding RTT estimation, which is much more precise than updating the previous estimation or the initialization value with each ACK. This feature comes with memory footprint increase. Given a connection with P paths, there would be P^2 RTTs to track. Nevertheless, with a low number of paths (*e.g.*, Wi-Fi and cellular) the complexity increase should be minimal.

For a better MP scheduling, we choose the one-path ACK strategy. However, this strategy still has an issue that the default one does not have: one of the intermediate nodes might silently discard one direction traffic, which could potentially terminate all paths except the ACK one [IIb, Issue 190[3]]. To avoid this, an ACK or a ping should be sent on the non-ACK paths. Combining both, each endpoint can have estimations of all possible RTTs. The receiving endpoint could send ACKs on non-ACK paths every ACK delay defined for each path, replicate the ACK on the ACK path after an arbitrary number of ACKs, or even do not send anything until one of the paths becomes inactive.

The most common example of MP use is a smartphone sending or receiving information through Wi-Fi and cellular networks. One of the MP QUIC use cases analyzed in IETF QUIC Working Group (WG) was video streaming from a smartphone using a Wi-Fi hotspot to aggregate the BW [LM20]. Low Earth Orbit (LEO) satellites can already directly connect to a smartphone for transferring text messages, with a more common internet traffic coming in the future [Lau23]. It is thus not too early to consider three paths for a smartphone. If available, a smartphone will first try to use Wi-Fi, which will surely be the first path on which the connection will be established. If the receiving endpoint does not implement the capacity to select an ACK path, then the ACKs would be sent on the first one, defined over Wi-Fi network. Path heterogeneity could result in different ACK frequencies on each path, triggering very frequent ACK packets transmissions over the shared access Wi-Fi link, with subsequent collisions with data packets going in the other direction. One way to mitigate this is adjusting ACK frequency on each path as specified in [ISK23]. The solution we choose is grouping ACKs from different paths in one packet.

[3] `https://github.com/quicwg/multipath/issues/190`, Accessed: 25th April, 2023.

When a path is ready to transmit an ACK, it will be delayed until all paths have an ACK to send. If one of the paths is too slow triggering its ACK, all delayed ACKs will be transmitted upon reaching the earliest maximum ACK delay. Paths recovering from a congestion do not delay their ACKs. However, if at the time of sending the ACK for the recovering path there are ACKs ready for transmission on other paths, these will be included in the ACK packet.

Given that ACKs can travel through different paths, it needs to specify the path whose packet numbers are acknowledged. [LMC+23] defines a new ACK_MP frame, same as normal ACK frame but with path CID field to identify the path. [LMC+23] recommends always using ACK_MP frames once MP extension has been negotiated. If an endpoint receives a regular ACK frame on any path, it is treated as it corresponded to the CID with the sequence number zero, *i.e.*, the first packet number space of the first path. In our proposal we replace the CID field in ACK_MP frame with the explicit path ID and change the interpretation of the regular ACK frame to better match the principle of maximal reuse of SP QUIC design: the regular ACK frame received on a path acknowledges the packets transmitted on that path.

The last adjustment in loss detection mechanisms is reporting FEC module the loss statistics, as was done for rQUIC (Chapter 3).

4.2.6 | Frame Handling

QUIC transmits control information and data in specialized frames. Stream data and control frames are managed by the stream manager block. Other control frames related to the connection are enqueued in a specialized control frame queue. On the Figure 4.1 we connect with the control frame queue the components that could generate a control frame. We keep control frames related to the path in the corresponding queue. However, some control frames intended for one path could be sent on another one, as in the case of ACK frames. To enable that, control frame queue has to be connected to MP manager.

Other frames defined outside MP extension could also be path specific, but for any reason would have to be enqueued in another path. To process these frames right, the receiving endpoint has to pass them to the corresponding path. In Section 5.4 we discuss a new frame that would enable treating any frame in the intended path.

Frame reading mechanism remains mostly the same, except that now it could read an ACK frame intended for another path. This component is extended to pass these frames to the corresponding paths.

4.3 | Path Scheduling

The implementation of MP as SP connections running in parallel offers the possibility to maximally utilize the throughput available to each path: a path will build a packet and pull data to fill it as soon as congestion and flow control allow it. However, if paths' one-way delays are very different, this strategy could send too many packets on the slowest path delaying the MP connection transmission time.

Most of existing MP schedulers decide which path should be the one to send a packet. If a supposedly optimal path has a very big CWND, such a scheduler could be choosing that path repeatedly, missing transmission opportunities on the other paths. In this section we describe the MP scheduler that fits our implementation. Since we focus on the interaction between MP and FEC extensions, the study comparing this scheduler with others will be carried after completing this thesis.

To maximize path utilization, the paths should be allowed to prepare a packet and fill it with application data as soon as congestion and flow control allow it. To minimize the transmission time on a MP connection, which equals the transmission time of the slowest path, we need to ensure this time is the same on all paths. To achieve that without sacrificing paths' BW utilization, we need to limit the total amount of data sent through each path. Wang *et al.* have already proposed and implemented this idea in [WGX19]. Despite the similarity with [WGX19], we reach slightly different results.

4.3.1 | Scheduler Inputs

To calculate transmission completion time we need the estimations of RTT and BW. As mentioned in Section 4.1, BW estimation is calculated with CWND, which covers both application and control data, including packets' and frames' headers. Packet number field and most of frames' fields are defined as variable integers, meaning the payload is not going to be a constant, even if the path Maximum Transmission Unit (MTU) is. In the absence of Planck constant in the context of transport layer protocols, we represent the throughput with $\hbar$. The throughput can be calculated as an adjustment to the estimated BW based on packet length and payload or overhead. Equation 4.1 shows its calculation based on payload length.

$$\hbar := BW \frac{payload\ length}{packet\ length} = \frac{5}{4} \frac{CWND}{RTT} \frac{payload\ length}{packet\ length} \tag{4.1}$$

The increase in the variable integers included in packet and stream headers is the same for each path and stream. Even if paths' BWs are very different from each other, the overhead difference of pure data packets will be very low: the packet number field

length ranges between 1 and 4 bytes; stream offset field, between 1 and 8 bytes. The difference between stream IDs should not be too big, but even if it was, the stream ID field length ranges between 1 and 8 bytes. In a very extreme case, the overhead difference could grow up to 17 bytes. IPv6 requires a minimum MTU of 1280 bytes [DH17]. IPv6 and UDP headers are 40 and 8 bytes long [DH17, Pos80], which leaves to a QUIC packet traversing an IPv6 network with minimal MTU a total of 1232 bytes. We consider that the overhead variations in data packets is negligible. Nevertheless, new extensions could introduce more overhead or explicitly limit the payload of stream frames. The combination of multiple extensions and very different path MTUs could make the throughput estimation non-negligible. In our experiments we only use one extension besides MP, the FEC. As we explain further in Section 4.4.2, our updated FEC introduces a constant overhead. We consider that in our experiments we can take the BW estimation as throughput.

Data throughput is also affected by control frames. Assuming that each path will transmit the same amount of control traffic, data traffic throughput will be reduced in the same proportion on all paths. To correctly estimate data distribution across paths we can ignore control frames. However, depending on the extensions in use and path MTUs, different paths could suffer throughput reductions of non-negligible difference.

QUIC treats all information as byte flows, not packet flows [IT21]. Thus, we work with the throughput in terms of bytes per time unit, not packets. FEC integration will require working with symbols instead of bytes.

4.3.2 | Data Limit for each Path

The transmission time on any path p should be the same as on the reference path r, which is expressed in equation 4.2.

$$T_{nxt(p)} + \frac{N_p}{\hbar_p} + (RTT_p - ACKdelay_p) = T_{nxt(r)} + \frac{N_r}{\hbar_r} + (RTT_r - ACKdelay_r) \tag{4.2}$$

Where $T_{nxt(p)}$ is time when the next packet will be transmitted on the path p, N_p is the number of bytes scheduled for the path p, and $\hbar_p$ is the average path throughput. If MP extension implements the default ACK transmission strategy, then $ACKdelay$ should be calculated as half of the corresponding RTT. With one-path ACK strategy this variable is the same for all paths and can be simplified.

The next transmission time T_{nxt} and RTT are path parameters that are not going to change during these calculations. They can be grouped into another variable, delivery delay D.

$$D \coloneqq T_{nxt} + RTT - ACKdelay \tag{4.3}$$

With these considerations, the equation 4.2 can be rewritten as follows.

$$D_p + \frac{N_p}{\hbar_p} = D_r + \frac{N_r}{\hbar_r} \tag{4.4}$$

Using one of the P paths as a reference leaves us a system of P equations of which only $P-1$ are linearly independent and P variables. The P^{th} linearly independent equation in the system is the relationship between the data sent through each path and the total data.

$$\sum_{i=1}^{P} N_i = N \tag{4.5}$$

We can find another expression for the sum of all N_i summing all of the equations defined as the equation 4.4. First, we clear the unknowns N_p.

$$N_p = \hbar_p \left(D_r - D_p + \frac{N_r}{\hbar_r} \right) \tag{4.6}$$

Summing the P equations we obtain the following equation where all unknowns are grouped like in equation 4.5 and only one unknown (N_r) outside the sum.

$$\sum_{i=1}^{P} N_i = \sum_{i=1}^{P} \hbar_i \left(D_r - D_i + \frac{N_r}{\hbar_r} \right) = \left(D_r + \frac{N_r}{\hbar_r} \right) \sum_{i=1}^{P} \hbar_i - \sum_{i=1}^{P} D_i \hbar_i \tag{4.7}$$

Given that we can use any path as a reference, we can rewrite the equation 4.7 for the path p instead of r. Equations 4.5 and 4.7 can be combined as follows.

$$\left(D_p + \frac{N_p}{\hbar_p} \right) \sum_{i=1}^{P} \hbar_i - \sum_{i=1}^{P} D_i \hbar_i = N \tag{4.8}$$

Clearing N_p we obtain the number of bytes N_p to be transmitted through the path p.

$$N_p = \hbar_p \left(\frac{N + \sum_{i=1}^{P} D_i \hbar_i}{\sum_{i=1}^{P} \hbar_i} - D_p \right) \tag{4.9}$$

If the total amount of data to transmit N is very small, some of the N_p could be zero or negative. This means that transmitting any packet on these paths will delay the transmission completion time. To find the correct information distribution, the equation 4.9 should be applied again, but only considering the paths with positive N_p. For other paths the N_p will be zero.

Wang *et al.* in 2019 proposed the same MP scheduling and combined it with a priority based stream scheduler for HTTP/2 application [WGX19]. After determining streams' priorities, their implementation was sending only one stream at a time. The overall transmission time of the streams meant to be transmitted in parallel was the same, but the completion time for each stream was minimal. As we discuss in Section 4.4.3, this approach can also simplify scheduling FEC redundancies.

To schedule the subsequent streams we can apply the equation 4.9, with the total number of bytes N corresponding to the next stream, and redefining the delay D. The new value is the one defined in equation 4.3 adding the already scheduled bytes divided by the throughput. Equation 4.10 represents the update of D for subsequent streams, with subindex p representing the path and s, the stream.

$$D_{p,s+1} = T_{nxt(p)} + RTT_p - ACKdelay_p + \frac{\sum_{i=1}^{s} N_{p,i}}{\hbar_p} \tag{4.10}$$

4.3.3 | Stream Management in MP

The proposal to transmit only one stream at a time presented in [WGX19] suits very well for the transmission of web objects. However, a web application could generate data for specific streams almost continuously. If the throughput suddenly decreases (*e.g.* CWND drop), the stream being transmitted would remain in the transmission queue for some time, delaying the transmission of other streams, which probably care delay sensitive data. This problem can also occur outside web traffic use-case.

Although streams were designed as an abstraction for web objects, they can be used to represent other things. Beyond the web traffic, there is a lot of interest in using QUIC in IoT environments [LSR+18, KD19, Egg20, FZG+21, DJ21, JFD+22, ALM22, SM23, IGK+23]. MQTT is an IoT application layer protocol that connects multiple devices through a *broker* [BBBG19]. Devices send their messages through a labeled stream called *topic* to the broker. Any device subscribed to that topic will receive this message from the broker. An advanced IoT protocol stack could use MQTT over QUIC mapping topics on QUIC streams. In this use-case sending one stream at a time till it has no more data could harm IoT devices' interoperability. Furthermore, this approach would apply to the *messages* sent through the streams, but not the streams. In web traffic streams represent the objects, the messages, however, in other environments it is worth distinguishing the two as different abstraction layers.

Despite the advantages of sending only one stream at a time, in our implementation we do not follow this approach, keeping the original Round Robin (RR) stream scheduler.

4.3.4 | In-Order Delivery

The more heterogeneous are the paths, the higher is the risk of overflowing the receive buffer with out-of-order data, ending up in HoLB, with explicit signaling to the sender to halt the transmission for a time. In [LLT+14] Li *et al.* have demonstrated the importance of receive buffers to maximize the use of the aggregated BW. They achieved significant throughput improvements in MPTCP with the use of FEC and pre-blocking warning mechanism, which triggers the transmission of redundant packet when there is a risk of HoLB. Other strategies to prevent HoLB consist in pre-reordering data packets in the scheduler to anticipate out-of-order delivery [SBL+13, KLM+14, SCW+18]. A generic MP implementation should not count on FEC, but on its own components and the ones defined in QUIC. In Section 4.3.2 we have calculated the data limits on a path for the whole MP connection, without ensuring flow control safety.

Using one-way delay and BW estimation we could predict packet arrival on each path. This information can be used to send the information scheduled on slower paths: the slow path that starts to build a new packet could skip the information that would be sent and delivered on faster paths before the packet it is building will reach the receiver.

Information skipping in QUIC is more complex than in TCP. In TCP there is only one information flow, but in QUIC there are as many flows (streams) as the users need. Moreover, we cannot simply estimate the number of bytes to skip and divide by the number of active sending streams, because some of the streams could be prioritized and treated differently by the stream scheduler. Note that the estimation of skipping distance does not depend directly on the stream priorities, but on the stream scheduler, which acts in a specific way based on the priorities.

In SP QUIC stream frames are multiplexed into one information flow at the time of preparing a new packet. In MP QUIC there are multiple instances building a packet, meaning that streams will have to merge into one information flow before reaching any path. On the other hand, if there are at least as many streams as paths, the best option to ensure information in-order delivery would be assigning a complete stream to a path like in [SWZL19], until the paths reach their transmission limit calculated in Section 4.3.2. Although stream awareness can benefit a MP connection [RHB18, SWZL19, SWZ+20], we focus on merging streams into one information flow, leaving more complex interactions between stream and MP schedulers for future works.

Multiplexing streams into one information flow does not mean that stream frames will be enqueued in a new buffer and treated like one continuous data block, sending segments that not necessarily match the underlying stream frames. Instead, each path would transmit the corresponding stream frame. When a path is building a packet and

requests data to fill its contents, MP manager would pull as many new stream frames as necessary from the stream manager till reaching the frame that would be delivered in order if sent on that path. For instance, if a slow path estimates that by the time its packet will reach the receiver N new packets will be built and delivered on other paths, the stream manager would pull $N + 1$ stream frames, and the scheduler would deliver the last of them to the path building the new packet.

The problem of pulling stream frames for faster paths is that QUIC payload is not constant, as discussed in Section 4.3.1. We can identify three cases when the packet data payload reduces: packet number field or stream frame offset field increases; path MTU shrinks; and control frames are packed in the packet along with the stream frame. Payload decrease results in splitting the already prepared stream frame into two, sending the first part in the next packet and managing the transmission of the remaining one in the next send opportunity, possibly on another path. If path MTU has increased and QUIC includes MTU discovery mechanism, the new MTU could result underutilized. Moreover, preparing a packet with larger payload will require packing more information than what is provided in the pre-built stream frame. Possible strategies to solve these issues could include limiting the size of the pre-built stream frames (limiting the goodput as well) to anticipate the reduction of payload, managing the increase of stream frames with data from the same stream, splitting stream frames and merging them with the next ones.

The goal of altering frame order consists in avoiding HoLB at the receiver. To achieve that, it is not strictly required to receive all of the frames in order. To prevent HoLB in MPTCP with limited receive buffer, Li *et al.* implemented a coding solution focusing on redundancy triggers rather than on a strict in-order packet delivery [LLT+14]. As long as the ordered data can be received and passed to the application before the receive buffer is full, HoLB is prevented. To ensure that, the information flow could be split into small chunks, which then are sent to the network without any reordering at frame level, only limited by the corresponding path limit for the chunk as given in the equation 4.9. If a path is too slow to send any packet with the data from the current chunk, it can pack the data from the next one, whose limits are calculated with the equation 4.10.

The size of the chunks can be calculated as a portion of the receive buffer, which could be inferred from the send window increases signaled with MAX_DATA frames. If these frames are lost or arrive out of order, the endpoint that receives them will keep the highest *Maximum Data* field [IT21]. Since MAX_DATA frames do not have sequence numbers, if one of these frames is lost, the inference would result in excessively big values. To avoid any confusion, the receiving endpoint could explicitly notify how big should the data chunks be. Matching the chunk size announced by the receiver should

not interfere with flow control. The sender defining a new data chunk should take the minimum between the MP connection flow send window and the chunk size announced by the receiver.

The main difference between chunking the multiplexed information flow and pulling the stream frames from the stream manager in advance is that the stream frames built from a data chunk are defined during packet preparation, greatly simplifying stream frames' management. Another difference is the interaction between the scheduler and the stream manager. Frame pulling requires almost no modification to the stream manager. In the case of pulling data chunks, the stream manager has to provide data chunks for each sending stream proportional to its stream scheduling policy. These chunks can be implemented as large stream frames. However, the compliance with stream scheduling policy will require increasing the interaction between MP and stream manager.

Data chunk definition depends on the maximum size announced by the receiver and the MP connection flow control. Over time, the receiver could announce a new chunk size. A sender should only define the chunks whose data will be transmitted in the next time period. In other words, the sender would be defining new chunks until each path has data to send in the next time period or the end of information flow is met. This time period could be the time between ACKs, but then if an ACK frame is lost, some of the paths that finished transmitting their share of the last defined data chunk would stall. This should not happen if the time period is one RTT. New ACK frames, connection flow control changes or new chunk size announcements would be used to update the limits of the already defined chunks, but only for those whose transmission has not started yet.

Apart from poor scheduling, data can arrive out of order due to packet erasures. Retransmitted frames arrive after the ones received in the first transmission. To minimize HoLB, MP scheduler must ensure that frames detected as lost are retransmitted as soon as possible. In our implementation lost frames handling is transparent to the scheduler. All frames related to streams and stream management are handled by the stream manager, which will enqueue frames marked as lost first. If MP scheduler tries to ensure in-order delivery, the priorization of lost stream related frames is already ensured by the stream manager. Application data related frames that are not specific to any stream are managed by a dedicated queue in the path manager, and its frames are pulled by the packet packer component bypassing the scheduler. When path specific frames are lost, their retransmission is carried out by the corresponding path. If the path becomes unavailable, the retransmission of path specific frames is useless. Sending path specific control frames on another path could benefit MP performance, but in the absence of its immediate need we do not implement this feature.

4.4 | FEC

In this section we review what changes need to be done to our FEC implementation since rQUIC development. The main goal of this review is to fit FEC to perform best with MP. Nevertheless, other practical aspects are also considered.

4.4.1 | FEC Adjustments

Our previous rQUIC design (Section 3) built coded packets after QUIC encrypted the source ones. Without specific adjustments, coded packets are invisible to the congestion control algorithm. In spite of rQUIC inserting only one coded packet per generation, we have seen that at high loss rates the overhead created by adaptive coding is not negligible, with potentially significant impact on network congestion events. As we discussed in Section 3.6.2, at 5% loss rate, rQUIC adaptive coding can consume more than 20% of bandwidth. Measurements from in-flight communications reveal loss rates of 7% [RNB+18]. We assumed that it was not necessary to update CC with coded packets if we insert only one coded packet per generation, since the impact would be minimal. However, if high loss rates are due to network congestion, 20% of traffic not accounted by CC would lead to an even greater congestion. We thus conclude that even if at fixed coding rates the impact of not reporting coded packets to the CC can arguably be assumed minimal, with adaptive coding rates the coded packets must count towards the CWND. FEC operations before encryption are, on the one hand, not compromising congestion control in any case, and, besides the simplify coded packets management, especially in MP environments.

After finishing the main experiments described in Section 3.6, we evaluated different values of BTO to see its impact on the transmission completion time (Section 3.6.5). We observed that at BTO values equal or slightly greater than RTT, the time to complete web download and bulk transfer greatly decreases. After carefully analyzing the interaction of rQUIC with CC, we conclude that such transmission time reduction could be a result of other mechanisms than FEC recovery. Delaying a packet for one RTT or even more could help FEC recovery, but generating an ACK could recover the missing packet with a retransmission in the same time. If the buffered packet is released after the timeout without recovering the preceding one, the buffer will also release all of the consecutive packets after the first released one. In the absence of the ACK signaling, the loss and the need to reduce the CWND, will cause the sender to keep on sending its data at the same rate, even if the loss that delayed the ACK was provoked by network congestion. When the packets are released, the receiver will acknowledge them. After receiving

the ACK, the sender will at first reduce the CWND for the observed loss, but then, quickly increase it because of the reception of the subsequent packets. This way the CWND does not reduce to the appropriate values if BTO values are big. We conclude that buffering hinders the work of CC by forcing it to work with outdated information, always delaying the "bad news". If there is any need to buffer the incoming packets, the BTO should be very small. Rather than buffering packets at the receiver, the sender should be able to delay retransmissions to let FEC recover the losses. Hence, in the updated FEC module we do not buffer received packets.

rQUIC, as well as other FEC implementations, is based on a generic coding scheme. Detailed use case analysis was outside the scope of rQUIC. In [MCM+22] Michel *et al.* have identified three main categories of FEC use cases, to which they applied a versatile coding scheme based on [CMBM20]. These use cases are bulk transfer, limited receive buffer and delay-constrained messaging.

In *bulk transfers* only tail losses increase Download Completion Time (DCT). Lost packets are quickly identified with frequent ACKs triggered by newer packets reaching the receiver. To detect and correct tail losses, QUIC sends probe packets with already transmitted content after a timeout [IS21]. Sending FEC packets without the timeout allows to either recover losses as soon as these packets are received, or at least trigger an acknowledgement, which would speed up the retransmission. We update rQUIC to follow the same approach. Coded packets are inserted at the end of a transmission.

Tail losses can also affect the streams. Even if after finishing a stream other streams still send data that trigger regular ACKs, the loss of the last packet in stream delays message delivery by 1 RTT (ACK + retransmission). A coded packet sent right after the one carrying the last frame in the stream could avoid incurring in this delay. The last stream frame marks the end of the coding window. The beginning of the coding window should be the first packet in flight, since all previous packets are either delivered or lost. The prediction of stream termination is possible only with the knowledge of how streams are scheduled. Advanced interactions with the stream scheduler are not covered by the scope of this study. The stream scheduler presented in [WGX19] could greatly simplify end of stream protection, although this approach is not considered for this implementation, as discussed in Section 4.3.3.

Protecting bulk transfers before they reach the end might seem useless. However, if receiver's memory is very small, packet loss could induce HoLB. Connection flow control gives a good estimation of the receive buffer's availability. Thus, another trigger for sending coded symbols should be reaching the end of the connection flow control window.

Any data transmission can be seen as a series of bulk transfers of different sizes, messages that an application is sending to the other endpoint. A *delay-constrained messaging* application, such as video streaming, sends short and frequent messages (frames). Without FEC, each frame's tail loss can be recovered either by regular retransmission triggered by the next frame, or with tail loss probing [IS21]. If conventional recovery schemes allow delivering the messages on time, FEC packets insertion at the end of each message is unnecessary. If tail loss recovery would delay message delivery beyond its deadline, coded packets protecting in-flight packets can speed up recovery process. In [MCM+22] the authors extend their implementation with delivery deadline awareness to further improve FEC scheduling.

In specific cases FEC might not be enough to meet a deadline because of limited BW. If the deadline is missed, what should a deadline awareness implementation do with the message? Cancel its stream or try to deliver it after other messages? Advanced stream management for time-sensitive traffic through application-defined priorities [SCQH19, CMS+23], deadline specification, and stream cancelling [SCQH19, CMS+23, CLJY22, ZGL+19] is attracting researchers' interest. Despite potential improvements that these techniques can yield, they are beyond the scope of this work.

When the last packets are lost, the corresponding ACK is triggered after reaching the Probe Timeout (PTO) since the transmission of the last packet [IS21]. After PTO is reached, QUIC repeats the last packets to either recover the loss of one of them, or to at least trigger an ACK reporting the losses. We replace the repeated packets with coded ones, allowing the receiver to recover any packet, not only the ones used for probing.

Prediction of packet loss periodicity and duration would result in a more precise estimation of the number of required coded packets. However, given the limited use of FEC, the impact of such models would be minimal. Taking into account that 85% of TCP flows observed between 2008 and 2019 are smaller than 100 kB [BJH+21], the models might not learn, for most of the cases, the loss pattern, probably not even encounter any loss at all. Nevertheless, a simple estimation of the maximum burst loss could be used to create sufficient redundancies by the end of the transmission if the burst happens again.

When an acknowledgement reports losses that FEC failed to recover, it is better to retransmit lost packets instead of sending more coded ones. When implementing an extension or any feature, it is important to bear in mind the goal that this feature pursues. The goal of using FEC is to reduce communication latencies, while increasing robustness, not to use FEC only because it is available. QUIC records all in-flight packets and, by design, it is ready to retransmit the frames from the lost packets. Retransmitting a packet instead of sending a new coded one not only reduces computational complexity

to build it, but also saves decoding operations at the receiver. Note that after sending N coded packets and detecting L packet erasures with $L > N$, the sender would need to retransmit only $L - N$ source packets, with exactly the same content as in the first transmission.

4.4.2 | Updated FEC Operations

With the adjustments described in Section 4.4.1, FEC only protects the last part of the communication. More specifically, FEC should be applied only to the packets that are in flight when connection's last packet is sent. We call this packet range *coding set*. It comprises both source and coded packets at the transmitter, similarly to the *payload set* specified in [AAB+18], which refers to the symbols of both types that reached the receiver. Nevertheless, other definitions of coding set could be evaluated, such as coding sets comprised of only source packets.

Given the new approach to implement FEC, most of the packets will not be protected. FEC signaling consumes bytes in packets, which is why we simply do not include it till reaching the end of the communication, the last batch of packets in flight, the coding set. To start processing packets as source symbols and insert FEC headers, we need to predict the beginning of the coding set. To do that, we compare the number of remaining packets with the source ones. Remaining packets can be calculated from the remaining bytes in the connection, dividing them by the source symbol payload size, which in its turn can be calculated subtracting packet header and source symbol header lengths from the packet length. We can calculate the number of source and coded packets in the coding set with the help of loss rate estimation. Equalling the number of redundancies to the number of expected losses we make sure the receiver gets enough packets to recover the lost ones.

To activate FEC, *i.e.*, to start treating new packets as source symbols and insert FEC headers, two conditions must be met. The first condition is that the number of the remaining packets is greater or equal to the source ones, so that FEC can protect every packet in the coding set. The second condition is that after the next ACK, there would be less remaining packets than source ones. This will ensure that FEC is not protecting packets that can be recovered by ARQ.

If the last expected ACK before the coding set is lost, it will be impossible to know if the packets that should have been acknowledged have been received or not. It could be worth adding these packets to the coding set. Thus, the coding set could be increased with a margin. This margin could be limited to only the number of packets between consecutive ACKs, or comprise the packets sent in one RTT for greater robustness to ACK loss.

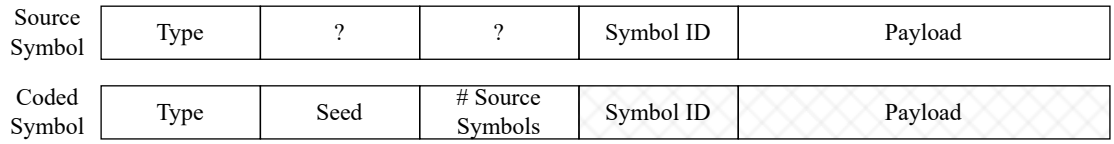


Figure 4.2: New FEC headers.

FEC headers change from rQUIC. We only signal protected and coded packets, the absence of FEC signaling is *per se* a signal that the packet is not protected. Source symbol header consists of frame type and symbol fields. It is placed at the beginning of the packet and the rest of the frames are considered its payload, the source symbol itself. Coded symbol header has the same fields plus the seed to generate coding coefficients and the number of source packets. Symbol ID and the payload fields of the coded symbols are linear combinations of the same fields of the source symbols. Both symbol formats are represented in Figure 4.2.

Source symbols have two less fields than coded ones which cannot be used to send more data, since it would increase the coded payload of the redundant symbols beyond the packet length. These two empty fields can be used to communicate extra information to the receiver, they can be omitted subtracting from the packet payload the two bytes, or source packets can be filled with two offset frames before source symbol so that source and coded packets are of the same length and indistinguishable after encryption.

The recovery of source packets should generate an ACK and a report on which packets have been recovered. Nevertheless, if the recovery was performed before receiving all of the source and coded packets, their reception must trigger another ACK. The unused fields in source symbols could carry the information about the number of source and coded symbols scheduled for the connection (path) where the symbol is sent. This information would help the receiver to better scheduler its ACKs

4.4.3 | Redundancies in MP

MP gives access to different networks with different loss patterns. A FEC extension can be applied to compensate losses on each path. However, in the presence of an ARQ mechanism that frequently sends ACKs, the coding windows will be small, where only one redundancy would be protecting against very high loss rates. If paths' packet loss rates are low but non-zero, a compound FEC could insert less redundancies protecting all paths than if it was protecting each path separately.

Extending FEC to P paths, we now have P coding sets CS, with source and redundant packets distributed among all paths. We can calculate their numbers with the loss

rate of each path. Equation 4.11 shows how the total amount of redundant packets R in all paths.

$$R = \left\lceil \sum_{i=1}^{P} CS_i \alpha_i \right\rceil \tag{4.11}$$

To make sure the coded packets protect all source ones, the former are sent through any path after transmitting the rest of packets. A path with a very high one-way delay could stop sending packets before other paths, not sending any coded packet at all. The number of redundancies sent on each path can be calculated by estimating the transmission time of the last R packets sent through any path. This allows calculating the number of source packets in every path, which is used to determine if a path should activate FEC or not.

Protecting packets from multiple paths can lead to situations when a packet sent on one path is recovered from the redundancy sent on another. If the recovered packet carried path specific control frames, the decoder will have to know the path for which these frames were intended. Thus, the symbol ID presented in Figure 4.2 must include the path ID. On the other hand, if FEC is applied to each path separately, the symbol ID does not need to include any path information.

FEC symbols can be scheduled as the rest of data, with the condition of not splitting the symbols. To achieve this, the MP scheduler has to be able to manage data in terms of symbols, and all of the FEC symbols should belong to the same data chunk. If paths have different MTUs, the smallest of them will be used, so that all source symbols are of the same length.

It has been proved that a stream aware MP scheduler [RHB18, SWZL19, SWZ+20] can significantly improve QUIC goodput. FEC could protect not only the end of a connection, but the end of a message avoiding possible 1-RTT recoveries, coinciding with the end of a stream if streams are representing web objects or video frames. A stream scheduler that transmits only one stream at a time (as proposed in [WGX19]) could help schedule FEC packets. Whether the stream scheduler sends only one stream at a time or uses a more general scheme based on priorities and deadlines, this research line is not in the scope of the present work. Our studies count on single-stream connections.

4.5 | Summary

In this section we have reviewed QUIC architecture and its extension to include MP and FEC functionality. We reviewed the new components in extended QUIC and the changes to the existing ones.

In the development of the MP extension we considered the strategy of replicating all connection related constructs surrounding the UDP socket. Although the idea is simple, its implementation raises many questions, such as: which path should be used to send ACKs; how to handle CIDs and packet number spaces; and how to schedule multiple streams on multiple paths. The ongoing design of a generic MP extension is not finished yet, with a lot of discussion taking place, aimed at solving the question we mentioned among many other issues. We address these discussions to better assess the design of our own MP extension. We also analyzed existing MP schedulers to select one that would best adapt to multiple connection managers running in parallel.

We updated our FEC design enabling its interactions with MP and improving its interaction with CC. We also addressed different FEC use cases at transport layer identified in [MCM+22]. At the time of writing, the assessment of the proposed design is ongoing.

5

Summary and Outline

In today's modern society it is hard to imagine not being connected to internet. Laptops, smartphones, tablets and other devices allow people to access any kind of information at almost any time. Modern automated remote control and monitoring applications are based on interactions between multiple devices connected through internet, leading to the appearance of specialized communication protocols and network architectures encompassed by the IoT paradigm. The growing global internet traffic and the use of new types of communications and links, especially those that are wireless and so error-prone, require improvements in network architectures and protocol stacks. QUIC is a new transport protocol originally developed to improve web traffic [LIB+17], although its design is not limited to any specific use case.

QUIC offers a functionality similar to TCP, bringing a series of improvements. Its transport parameters enable an easy configuration for specific connections. Its user-space implementation and a version negotiation [SR22] make QUIC easy to customize. Improved connection establishment and loss recovery mechanisms reduce latencies. To further reduce them, we seek to implement extensions that would enable such improvements. More specifically, we focus on FEC, to minimize recovery latencies, and on MP, to deliver data through different networks and complete transmissions in less time.

In this chapter we first summarize the research carried out during this PhD thesis, discussing the most relevant results and drawing conclusions. Then we explore the research lines that have been opened after the work discussed herewith, which would lead to further latency reductions. At the time of writing, the assessment of such proposed design is ongoing.

5.1 | FEC

There are many coding techniques, potentially exploited at the transport layer, which can significantly improve a communication suffering from data corruption or loss. Nevertheless, applying a coding technique to a reliable protocol capable of recovering data losses through ARQ mechanisms can result too complex for the low latency reduction that can be achieved.

Before extending an existing transport protocol, we analyzed existing coding techniques and evaluated those that could best suit a protocol with an ARQ mechanism. We evaluated convolutional systematic RLC for generation sizes of 64 and 256 source symbols with different number of redundant symbols. We defined a special condition for the convolutional coding: each source symbol should be protected by the same number of coded symbols. We observed, in a simulation-based study, that convolutional coding was able to reduce the recovery latency proportionally to the number of concurrent generations. Apart from the latencies, we also studied the required overhead, defined as the ratio between the number of coded symbols and all transmitted symbols, both source and coded. More specifically, we observed the saturation overhead, the one at which full message reception probability reaches its maximum. Results evince that minimizing saturation overhead optimizes throughput, but not the delay. This observation is especially important in those cases where the throughput requirement is more relevant than the delay.

While implementing coding techniques in QUIC protocol, the first thing we found was that ACK frequency used by the implementation we chose is the one recommended in [IT21, Section 13.2.2], i.e. every two packets. Neither the generation sizes of 64 symbols nor convolutional coding would significantly reduce recovery latency, because by the time the coded symbols are received, ARQ has already recovered most of the losses. Our coding scheme consisted in inserting only one coded symbol for each generation of variable size. Generation size was determined by the adaptive coding rate presented in [FKCA18]. All coding operations were performed over packets ready to be sent to the network. To avoid unnecessary retransmissions, we delayed ACK generation at the receiver by buffering packets arrived out of order. If the order was altered due to a packet loss, waiting for a coded packet within a reasonable time can indeed prevent the retransmission and so reduce the recovery latency despite the waiting time. Nevertheless, the packets should not stay for too long in the buffer. We defined BTO after which packets are released as QUIC maximum ACK delay minus a certain margin. A buffered packet that reaches its BTO is released from the buffer to be processed by QUIC as usual, triggering the corresponding ACK within the margins established during QUIC handshake.

We evaluated our implementation using ns-3 simulator emulating Wi-Fi, mobile and satellite links between the endpoints with packet loss rates between 0 and 5%. We used three traffic patterns: bulk transfer, small web page download and video streaming. Simulation results showed significant improvements in the presence of packet losses but for a particular exception in video streaming scenario. Because of FEC overhead, the QoE and the probability of reaching higher resolutions at 1% loss using Wi-Fi link were better for the original QUIC. Video streaming QoE improvement on satellite link was minimal, due to high RTT. After finishing the main experiments, we also evaluated different values of BTO to see its impact on the transmission completion time.

To enable FEC and MP interaction, our FEC module needed to be updated. We carefully analyzed our first implementation and the most recent related works with other FEC extension proposals. The first changes we introduced addressed CC interaction: coded packets now count towards CWND and received packets are no longer buffered to give FEC more opportunities to recover losses. Following the analysis of FEC use cases for different traffic types published in [MCM+22], we adopt the idea of protecting only the end of a message and the end of the send window (protecting the receive buffer). If a packet can be recovered with ARQ before the message transmission concludes, the transmission time perceived by the application will be the same as if the loss was recovered by FEC. At the same time, ARQ sends only one packet for each erasure, unlike FEC, which in its attempts to predict a loss can send several coded packets. On the other hand, if one of the last packets of the transmission is lost, its recovery can take approximately one RTT (ACK and retransmission), a time that FEC can save. Furthermore, we also reviewed how multiple paths should be protected with FEC.

5.2 | Multipath

To assess the joint performance of FEC and MP, we needed to add the MP to our implementation. To address the challenges that arise from handling multiple sockets, we ended up implementing our own MP module. In what follows we summarize the most relevant challenges we found.

- Path scheduling.
  Most of existing schedulers choose one path or another for an already prepared packet. If a path is chosen several times in a row, the other path looses sending opportunities. We thus allow each path to decide when to send a packet and send them as soon as path's congestion and flow controls allow it. This kind of scheduler would need to limit the data sent on each path to ensure the minimal

transmission time. We reviewed the scheduling algorithms based on this principle and proposed our own version of such algorithms.

- Data source – stream selection.
  Whenever a path gets a transmission opportunity and starts 'preparing' a new packet, it needs to choose the source from which it will take new data to send. In QUIC there are as many data sources as send streams. In SP QUIC, stream data are multiplexed into one data flow at the connection level, when the only existing path pulls data from the stream manager. Leaving stream-to-path scheduling outside this thesis' scope, we perform this multiplexing at the MP manager.

- Receive buffer overflow.
  Data frames could be delivered to QUIC connection out of order due to path heterogeneity. In these cases QUIC connection could suffer HoLB: if the receive buffer is filled with out-of-order data, it will not be able to receive new data and so stall the communication on all paths. Sending only one or few streams at a time will not necessarily avoid HoLB, because the amount of data to send on the stream could be larger than the receive buffer. The existing solutions include altering packet order at the sender and using FEC. For MP extension alone, we propose sending the multiplexed data flow in chunks that would fit the MP connection flow limits. Within each chunk, the data can be sent in any order.

- Scheduling control frames.
  QUIC sends its control signals in specialized frames in packet payloads, along with data frames. Reviewing scheduling algorithms, we had to distinguish between path specific and data related frames. We concluded that data related frames, *i.e.*, stream frames and frames related to the control of one or multiple streams, should be multiplexed into the unified data flow. Path specific frames, such as flow control frames, should be sent on the path to which they are related.

- Packet acknowledgement.
  We reviewed two strategies to acknowledge packets: acknowledge on the path where the packets were sent, or use only one path to send the ACKs for packets from any path. We discussed the differences and advantages of both strategies.

- Path identification.
  According to the existing draft specification of IETF MP extension for QUIC, each path is identified by its CID sequence number [LMC+23]. This results in several

design complications that can be solved by assigning each path an explicit identification. We discuss the differences of both proposals and explain why we choose explicit path IDs.

Analyzing frame scheduling, we identified two types of frames: data related and path specific. This classification translates in selecting the queue that would manage a frame. To enable MP compatibility, other extensions should indicate to which type do their frames belong. QUIC datagram extension [PKS22] introduces a new datagram frame, which is of data related type. ACK frequency extension introduces two new frames [ISK23], both being path specific. Our FEC extension introduces two frames: source and coded symbols. Source symbols are path specific, and they include a header to indicate that the rest of the packet is FEC protected. If FEC is applied to each path separately, then coded symbol is a path specific frame. If FEC is applied to all paths, then coded symbols can be seen as data frames that are sent at specific moments.

As mentioned earlier, packet acknowledgements corresponding to one path can be sent on another one. QUIC original ACK frames in our extension are path specific, they are sent on the same path whose packets are being acknowledged. However, the new MP ACK frame introduced in [LMC+23] is a special case. It is explicitly not path specific, because in our implementation it is used to signal packet acknowledgements from another path. It can neither be viewed as a data related frame, because it cannot be added to the unified data flow and wait to be transmitted by an arbitrary path, it has to be sent immediately on the path indicated by an ACK scheduling policy. The MP ACK frame is specific to MP extension, which modifies its management. We can conclude that the suggested frame classification into data related and path specific frames applies to the frames that do not modify MP behaviour.

5.3 | Multistreaming

One of the most distinctive QUIC novelties compared to TCP is the possibility to handle multiple data flows or *streams* within one connection. QUIC specification defines streams as information flow abstractions which the implementations should be able to prioritize [IT21]. Applications like video streaming would make a greater use of streams if they could include more control elements, such as deadlines and an advanced stream cancelling mechanism [SCQH19, CMS+23, CLJY22, ZGL+19].

We observed that combining streams without the advanced features with MP and FEC extensions already poses interesting challenges. Although this work does not study the issue of scheduling streams, it is worth reviewing the interactions between stream

manager and the extensions implemented in this work. Other QUIC extensions could have similar interactions with the stream manager.

In MP there are many ways to schedule an information flow through different paths. To handle multiple streams we multiplex them before MP scheduling. However, it is possible to optimize stream completion time designing scheduling algorithms comprising both streams and paths [RHB18, SWZL19, WGX19, SWZ+20]. MP schedulers that alter packet order to adapt to path heterogeneity, as the ones proposed in [SBL+13, KLM+14, SCW+18], would need to pull certain stream frames skipping others. To provide the appropriate stream frames to each path, stream scheduler would need to know which path is trying to pull more data from it, its throughput and delay estimations, and the MTU. Moreover, the stream scheduler could distribute stream frames among paths based on their throughput and one-way delay. Even in SP connections path information could help the stream scheduler to adjust its scheduling to deliver the streams before the specified deadlines.

We conclude that stream schedulers should consume path related information and provide each path stream frames. If MP manager uses its own MP scheduler, it should be able to communicate the stream manager the path that is trying to pull new data. We believe that path related extensions should provide the stream manager information regarding the impact they have on a path, either through a (multi)path manager or directly to the stream scheduler. Such information could be the overhead generated by extension's frames and the throughput modification.

FEC protection applied at the end of a stream can save up to one RTT in stream completion time. To achieve that, FEC could be applied separately to every sending stream, which might be inefficient. Especially in the presence of multiple streams multiplexed over time, the older stream frames protected by FEC will be already acknowledged, and those that are lost will be recovered with ARQ mechanism. Such frames would unnecessarily increase FEC operations. A FEC implementation that sends coded packets at the end of a stream would need to predict when the last stream frame would be sent, to protect only packets in flight. This prediction is only possible knowing how streams are scheduled. A FEC aware stream scheduler would activate FEC based on bytes in flight, its own stream scheduling and streams' remaining bytes. A more generic stream manager would not be aware of all of the extensions interacting with it. To interact with FEC, the stream manager would need to limit its frames to a specific symbols size provided by the FEC module, report the bytes in flight, and report if any stream is going to finish within the encoding window provided by FEC.

5.4 | Future Research Lines

5.4.1 | FEC

FEC and ARQ are designed to solve the same issue: recovering lost information. ARQ is the mechanism that introduces minimal overhead, for each lost packet it generates only one retransmission. If ARQ feedback is sufficiently frequent, FEC packets sent throughout a communication become useless very fast, since most of the losses are retransmitted via ARQ. An extension modifying ACK frequency could result in sufficiently spaced ACKs to give FEC more recovery opportunities. Thus, it is worth exploring the combination of FEC and reduced ACK frequency.

Another way FEC can be useful within a connection is delaying retransmissions of lost frames till the expected time of coded packets' reception. Its combination with ACK frequency variations can potentially result in an efficient FEC application, avoiding early retransmissions and too long ACK delays.

One practical aspect of using an extension in any environment is to stop using it if the expected improvement is not achieved (improvement-complexity trade-off), especially when the extension is worsening the overall performance. If FEC extension is enabled in the middle of a communication, it could be outperformed by ARQ, generating redundancies that will never be used. It is thus important to include a mechanism to evaluate FEC performance and disable the extension when it is no longer beneficial.

QUIC datagram extension enables unreliable transmissions. However, depending on application's needs, it could be worth making a vague attempt to recover lost datagrams with FEC.

Congestion events can be expected when the RTT increases, probably due to an intermediate node becoming congested, or when CWND approaches the value when a packet was lost and the CWND had to drop. A careful congestion prediction could be used to temporarily activate FEC, thus preventing the retransmissions due to congestion losses.

5.4.2 | MP

When FEC is protecting the packets only at the end of a message, the coding window on each path corresponds to the packets in-flight. These windows are easy to combine for a joint FEC protection of the whole MP connection. However, if FEC could be used in the middle of a connection, synchronizing coding windows of each path becomes challenging. Not only ACK frequency variation could impact MP ACK transmission strategy,

but also protecting packets from a slower path could delay decoding operations. Apart from analyzing the coding windows, a MP sender would have to discard paths for the joint protection based on the estimated packet arrivals.

MP scheduling that alters information order to adapt to path heterogeneity could end up packing two or more non-consecutive data frames from the same stream. To save the resulting overhead, the STREAM frame could be extended with ranges similar to ACK frame ranges. The first STREAM frame would include the offset and length fields. Instead of inserting the next frame with its full header, we propose to extend the header of the first frame with a field indicating the number of grouped frames and the offset of the next frame. The latter would be expressed as the length of the gap preceding it. If there is a third frame to append, the compound header would include the length of the second frame before adding the offset of the third one. This new compound frame adds one new field for the number of appended steams, and saves the type and stream ID fields for all the subsequent frames; it also reduces the offset field size. We did not implement this frame because of the small goodput gain. However, if due to losses and reordering in MP there were multiple data frames corresponding to the same stream and the MTU of the path where the new packet is going to be sent is very low, using extended stream frames could improve path goodput.

New QUIC extensions could add connection control frames, which in the context of MP would be path specific. If their delivery were urgent, and the corresponding path were suffering a severe congestion, control frames intended for one path should be sent on another one. To indicate the receiving endpoint that the frame specified outside MP extension is meant for another path, we propose the INTENDED_PATH frame. It has only two variable integer fields: frame type and the destination path ID. This new frame flags that all following frames are intended to be processed by the path specified in the path ID field.

In our MP extension we simplify stream to path scheduling by multiplexing all data and data related (control) frames into one data flow. However, a MP aware stream manager could improve ordered frame delivery and reduce stream completion time [RHB18, SWZL19, WGX19, SWZ+20]. Stream extended with additional control, such as deadlines, prioritization and partial reliability with a safe stream cancellation [SCQH19, CMS+23, CLJY22, ZGL+19], could benefit from MP awareness, especially for scheduling time critical streams.

5.4.3 | IoT

The existing and upcoming IoT use cases could greatly benefit from an extended QUIC. IoT communications could benefit from MQTT over QUIC [KD19, FZG+21]. Despite QUIC streams were intended to represent web objects [LIB+17], MQTT topics can be mapped to QUIC streams. QUIC can be run on traffic aggregators in IoT environments. However, after undergoing sufficient optimizations, QUIC could also run on low-end devices. In [Egg20], Lars Eggert evaluated QUIC in two 32-bit IoT edge devices. By simplifying QUIC functionality, he was able to fit a QUIC client in a few tens of kilobytes.

Nevertheless, QUIC can be run on more advanced devices with a relatively powerful processor onboard. The growing use of unmanned vehicles and Artificial Intelligence (AI) along with mobile networks evolution foster new IoT applications. One of these applications could be a system for rescue missions based on different drone swarms guided by AI, with each swarm being specialized on different tasks. One swarm would monitor weather conditions, another would overfly the area collecting data for identifying survivors and finding transportation routes, ground-based and aerial swarms would clean the debris, etc. One of the drones could aggregate the traffic from other drones and forward it to a remote server using a QUIC connection, mapping each data flow to a stream. In this scenario, the traffic aggregating drone, especially if it has multiple network interfaces, can benefit from the extensions studied in this work and the research lines analyzed in this section.

5.5 | Thesis Contributions

This thesis resulted in multiple contributions to the state of the art. Some of them (J1, C1, C2, C3, O1), were directly derived from the research carried out within the scope of this thesis, while others (J2, C4, C5) were collaborations that were opened thanks to the expertise and knowledge that was acquired during this period, which allowed to support other research activities with some common goals.

5.5.1 | Journals

J1 **Mihail Zverev**, Pablo Garrido, Fátima Fernández, Josu Bilbao, Özgü Alay, Simone Ferlin, Anna Brunström, and Ramón Agüero. Robust QUIC: Integrating Practical Coding in a Low Latency Transport Protocol. *IEEE Access*, 9:138225–138244, 2021.

J2 Fátima Fernández, **Mihail Zverev**, Pablo Garrido, José R. Juárez, Josu Bilbao, and Ramón Agüero. Even Lower Latency in IIoT: Evaluation of QUIC in Industrial IoT Scenarios. *Sensors*, 21(17), 2021.

5.5.2 | Conferences

C1 **Mihail Zverev**, Pablo Garrido, Ramón Agüero, and Josu Bilbao. Systematic Network Coding with Overlap for IoT Scenarios. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Barcelona, Spain, pages 1–6, 2019.

C2 **Mihail Zverev**, Pablo Garrido, Ramón Agüero, and Josu Bilbao. Combinación de Network Coding Sistem'atico y Solapamiento en escenarios IoT. In *XIV Jornadas de Ingeniería Telemática (JITEL 2019)*, Zaragoza, Spain, pages 1–6, 2019.

C3 **Mihail Zverev**, Pablo Garrido, Ramón Agüero, and Josu Bilbao. Network Coding for IIoT Multi-Cloud Environments. In *Proceedings of the 9th International Conference on the Internet of Things*, Bilbao, Spain, 2019.

C4 Fátima Fernández, **Mihail Zverev**, Pablo Garrido, José R. Juárez, Josu Bilbao, and Ramón Agüero. And QUIC meets IoT: performance assessment of MQTT over QUIC. In *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Thessaloniki, Greece, pages 1–6, 2020.

C5 Fátima Fernández, **Mihail Zverev**, Pablo Garrido, José R. Juárez, Josu Bilbao, and Ramón Agüero. Evolución del Stack IoT: MQTT sobre QUIC. In *XV Jornadas de Ingeniería Telemática (JITEL 2021)*, A Coruña, Spain, pages 1–8, 2021.

5.5.3 | Other Contributions

O1 Presentation and discussion of rQUIC FEC/NC extension for QUIC at Coding for efficient NetWork Communications Research Group (NWCRG), 105th IETF meeting, 2019.

References

[AAB+18] Brian Adamson, Cédric Adjih, Josu Bilbao, Victor Firoiu, Frank Fitzek, Samah A. M. Ghanem, Emmanuel Lochin, Antonia Masucci, Marie-Jose Montpetit, Morten V. Pedersen, Goiuri Peralta, Vincent Roca, Paresh Saxena, and Senthil Sivakumar. Taxonomy of Coding Techniques for Efficient Network Communications. RFC 8406, June 2018.

[ACC+07] G Albertazzi, M Chiani, G E Corazza, A Duverdier, H Ernst, W Gappmair, G Liva, and S Papaharalabos. *Forward Error Correction*, pages 117–174. Springer US, Boston, MA, 2007.

[ALM22] Ahmed Alqattaa, Daniel Loebenberger, and Lukas Moeges. Analyzing the latency of quic over an iot gateway. In *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6, 2022.

[AMD20] Wei An, Muriel Médard, and Ken R. Duffy. Keep the bursts and ditch the interleavers. *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, nov 2020.

[ANLY00] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, jul 2000.

[APA19] Khaled Alferaidi, Robert Piechocki, and F Alfordy. Improving TCP Performance in Multi-Hop Coded Wireless Networks. In *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, pages 1–6, may 2019.

[ATP20] Vipindev Adat Vasudevan, Christos Tselios, and Ilias Politis. On Security Against Pollution Attacks in Network Coding Enabled 5G Networks. *IEEE Access*, 8:38416–38437, 2020.

[BBBG19] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. MQTT version 5.0. Standard, Organization for the Advancement of Structured Information Standards (OASIS), 2019.

[Beg10] Ali C. Begen. RTP Payload Format for 1-D Interleaved Parity Forward Error Correction (FEC). RFC 6015, October 2010.

[BFPT99] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based error control for Internet telephony. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 3, pages 1453–1460 vol.3, 1999.

[BJH+21] Simon Bauer, Benedikt Jaeger, Fabian Helfert, Philippe Barias, and Georg Carle. On the Evolution of Internet Flow Characteristics. In *Proceedings of the Applied Networking Research Workshop*, ANRW '21, page 29–35, New York, NY, USA, 2021. Association for Computing Machinery.

[Bra89a] Robert T. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, October 1989.

[Bra89b] Robert T. Braden. Requirements for Internet Hosts - Application and Support. RFC 1123, October 1989.

[CB18] Quentin De Coninck and Olivier Bonaventure. Multipath Extension for QUIC. Internet-Draft draft-deconinck-quic-multipath-01, Internet Engineering Task Force, 9 2018. Work in Progress.

[CCA+17] Kae Won Choi, Young Su Cho, Aneta, Ji Wun Lee, Sung Min Cho, and Jaehyuk Choi. Optimal load balancing scheduler for mptcp-based bandwidth aggregation in heterogeneous wireless environments. *Computer Communications*, 112:116–130, 2017.

[CKZC19] Federico Chiariotti, Stepan Kucera, Andrea Zanella, and Holger Claussen. Analysis and design of a latency control protocol for multi-path data delivery with pre-defined qos guarantees. *IEEE/ACM Transactions on Networking*, 27(3):1165–1178, 2019.

[CLJY22] Jianping Chen, Tianyi Liu, Junxian Jing, and Yongyi Yu. An Unreliable Extension to QUIC. Internet-Draft draft-chen-quic-quicu-01, Internet Engineering Task Force, September 2022. Work in Progress.

[CM03] Z. Cataltepe and P. Moghe. Characterizing nature and location of congestion on the public Internet. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, pages 741–746 vol.2, 2003.

[CMBM20] Alejandro Cohen, Derya Malak, Vered Bar Bracha, and Muriel Médard. Adaptive causal network coding with feedback. *IEEE Transactions on Communications*, 68(7):4325–4341, 2020.

[CMS+23] Yong Cui, Chuan Ma, Hang Shi, Kai Zheng, and Wei Wang. Deadline-aware Transport Protocol. Internet-Draft draft-shi-quic-dtp-07, Internet Engineering Task Force, January 2023. Work in Progress.

[CWW+15] Yong Cui, Lian Wang, Xin Wang, Hongyi Wang, and Yining Wang. FMTCP: A fountain code-based multipath transmission control protocol. *IEEE/ACM Transactions on Networking (ToN)*, 23(2):465–478, 2015.

[CYX+09] Alix LH Chow, Hao Yang, Cathy H Xia, Minkyong Kim, Zhen Liu, and Hui Lei. EMS: Encoded multipath streaming for real-time live streaming applications. In *2009 17th IEEE International Conference on Network Protocols*, pages 233–243. IEEE, 2009.

[DB21] Quentin De Coninck and Olivier Bonaventure. Multipath Extensions for QUIC (MP-QUIC), May 2021. Work in Progress.

[DCB17] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 160–166, New York, NY, USA, 2017. Association for Computing Machinery.

[DCMP+19] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. Pluginizing QUIC. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 59–74, New York, NY, USA, 2019. Association for Computing Machinery.

[DH17] Dr. Steve E. Deering and Bob Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, July 2017.

[DJ21] Jasenka Dizdarević and Admela Jukan. Experimental benchmarking of http/quic protocol in iot cloud/edge continuum. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[Edd22] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, August 2022.

[Egg20] Lars Eggert. Towards securing the internet of things with quic, 2020.

[FAMB16] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 431–439, 2016.

[FKCA18] Simone Ferlin, Stepan Kucera, Holger Claussen, and Özgü Alay. MPTCP Meets FEC: Supporting Latency-Sensitive Applications Over Heterogeneous Networks. *IEEE/ACM Transactions on Networking*, 26(5):2005–2018, oct 2018.

[FLKC18] Kariem Fahmi, Douglas Leith, Stepan Kucera, and Holger Claussen. Low delay scheduling of objects over multiple wireless paths, 2018.

[FRH+20] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684, March 2020.

[FZG+21] Fátima Fernández, Mihail Zverev, Pablo Garrido, José R. Juárez, Josu Bilbao, and Ramón Agüero. Even lower latency in iiot: Evaluation of quic in industrial iot scenarios. *Sensors*, 21(17), 2021.

[GAF18] Frank Gabriel, Javier Acevedo, and Frank HP Fitzek. Network Coding on Wireless Multipath for Tactile Internet with Latency and Resilience Requirements. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[Gar18] Pablo Garrido Ortiz. *Opportunistic Network Coding over Wireless Networks*. PhD thesis, University of Cantabria, September 2018.

[GNM+17] Yihua Ethan Guo, Ashkan Nikravesh, Z. Morley Mao, Feng Qian, and Subhabrata Sen. Demo: Dems: Decoupled multipath scheduler for accelerating multipath transport. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 477–479, New York, NY, USA, 2017. Association for Computing Machinery.

[GNPS19] Yunmin Go, Hyunmin Noh, Goeon Park, and Hwangjun Song. Energy-Efficient HTTP Adaptive Streaming with Hybrid TCP/UDP Over Heterogeneous Wireless Networks. In *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–10, 2019.

[GSF+19] Pablo Garrido, Isabel Sanchez, Simone Ferlin, Ramon Aguero, and Özgü Alay. rQUIC: Integrating FEC with QUIC for Robust Wireless Communications. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, dec 2019.

[GSKL17] Andres Garcia-Saavedra, Mohammad Karzand, and Douglas J Leith. Low delay random linear coding and scheduling over multiple interfaces. *IEEE Transactions on Mobile Computing*, 16(11):3100–3114, 2017.

[GWP+18] Frank Gabriel, Simon Wunderlich, Sreekrishna Pandi, Frank H P Fitzek, and Martin Reisslein. Caterpillar RLNC With Feedback (CRLNC-FB): Reducing Delay in Selective Repeat ARQ Through Coding. *IEEE Access*, 6:44787–44802, 2018.

[HISW16] Ryan Hamilton, Jana Iyengar, Ian Swett, and Alyssa Wilk. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-hamilton-quic-transport-protocol-00, Internet Engineering Task Force, 07 2016. Work in Progress.

[HKM+03] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory, 2003. Proceedings.*, page 442, 2003.

[HMK+06] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michalle Effros, Jun Shi, and Ben Leong. A Random Linear Network Coding Approach to Multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[HNR+11] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend TCP? *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pages 181–194, 2011.

[HPFL09] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and T. Larsen. Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes. In *2009 IEEE International Conference on Communications Workshops*, pages 1–6, 2009.

[HS02] Hung-Yun Hsieh and R. Sivakumar. pTCP: an end-to-end transport layer protocol for striped connections. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 24–33, 2002.

[Hui21a] Christian Huitema. One-way delays for multipath quic. `https://huitema.wordpress.com/2021/09/19/one-way-delays-for-multipath-quic/`, Sep 2021. Accessed: 25th April, 2023.

[Hui21b] Christian Huitema. QUIC Multipath Negotiation Option. Internet-Draft draft-huitema-quic-mpath-option-01, Internet Engineering Task Force, September 2021. Work in Progress.

[Hui22] Christian Huitema. Quic Timestamps For Measuring One-Way Delays. Internet-Draft draft-huitema-quic-ts-08, Internet Engineering Task Force, August 2022. Work in Progress.

[HWCK07] Zheng Huang, Xin Wang, Xueqing Chen, and Haibin Kan. Network Coding with Interleaving. *Proceedings of the International Conference on Parallel Processing Workshops*, pages 1–6, 2007.

[IGK+23] Faheem Iqbal, Moneeb Gohar, Hanen Karamti, Walid Karamti, Seok-Joo Koh, and Jin-Ghoo Choi. Use of quic for amqp in iot networks. *Comput. Netw.*, 225(C), apr 2023.

[IIa] IETF QUIC WG and Internet Community. IETF QUIC WG Mail Archive. `https://mailarchive.ietf.org/arch/browse/quic/`. Accessed: 25th April, 2023.

[IIb] IETF QUIC WG and Internet Community. MP QUIC Github issues. `https://github.com/quicwg/multipath/issues`. Accessed: 25th April, 2023.

[II94] ISO and IEC. Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. Standard, International Organization for Standardization, Geneva, CH, November 1994.

[II19] ISO and IEC. Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. Standard, International Organization for Standardization, Geneva, CH, December 2019.

[IRB+11] Jana Iyengar, Costin Raiciu, Sebastien Barre, Mark J. Handley, and Alan Ford. Architectural Guidelines for Multipath TCP Development. RFC 6182, March 2011.

[IS21] Jana Iyengar and Ian Swett. QUIC Loss Detection and Congestion Control. RFC 9002, May 2021.

[ISK23] Jana Iyengar, Ian Swett, and Mirja Kühlewind. QUIC Acknowledgement Frequency, March 2023. Work in Progress.

[IT20] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport, June 2020. Work in Progress.

[IT21] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet Requests for Comments, May 2021.

[JB22] Farinaz Jowkarishasaltaneh and Jason But. An analysis of mptcp congestion control. *Telecom*, 3(4):581–609, 2022.

[JFD+22] Sidna Jeddou, Fátima Fernández, Luis Diez, Amine Baina, Najid Abdallah, and Ramón Agüero. Delay and energy consumption of mqtt over quic: An empirical characterization using commercial-off-the-shelf devices. *Sensors*, 22(10), 2022.

[KCP+13] MinJi Kim, Jason Cloud, Ali ParandehGheibi, Leonardo Urbina, Kerim Fouli, Douglas Leith, and Muriel Medard. Network Coded TCP (CTCP), 2013.

[KD19] Puneet Kumar and Behnam Dezfouli. Implementation and analysis of QUIC for MQTT. *Computer Networks*, 150:28–45, feb 2019.

[KGPL13] Ramin Khalili, Nicolas Gast, Miroslav Popovic, and Jean Yves Le Boudec. MPTCP is not pareto-optimal: Performance issues and a possible solution. *IEEE/ACM Transactions on Networking*, 21(5):1651–1665, 2013.

[KGPS14] Oh Chan Kwon, Yunmin Go, Yongseok Park, and Hwangjun Song. MPMTP: Multipath multimedia transport protocol using systematic raptor codes over wireless networks. *IEEE Transactions on Mobile Computing*, 14(9):1903–1916, 2014.

[KKR12] Dina Katabi, Sachin Katti, and Hariharan Rahul. *Chapter 2 - Harnessing Network Coding in Wireless Systems*, pages 39–60. Academic Press, Boston, 2012.

[KLL17] Bruno Yuji Lino Kimura, Demetrius C. S. F. Lima, and Antonio Alfredo Ferreira Loureiro. Alternative scheduling decisions for multipath tcp. *IEEE Communications Letters*, 21:2412–2415, 2017.

[KLM+14] Nicolas Kuhn, Emmanuel Lochin, Ahlem Mifdaoui, Golam Sarwar, Olivier Mehani, and Roksana Boreli. Daps: Intelligent delay-aware packet scheduling for multipath transport. In *2014 IEEE international conference on communications (ICC)*, pages 1222–1227. IEEE, 2014.

[KLMW22] Nicolas Kuhn, Emmanuel Lochin, François Michel, and Michael Welzl. Forward Erasure Correction (FEC) Coding and Congestion Control in Transport. RFC 9265, July 2022.

[KPUM12] MinJi Kim, Ali ParandehGheibi, Leonardo Urbina, and Muriel Meedard. CTCP: Coded TCP using multiple paths. *arXiv preprint arXiv:1212.1929*, 2012.

[KTK15] Krishnaprasad K, Mohit P. Tahiliani, and Vinay Kumar. TCP Kay: An end-to-end improvement to TCP performance in lossy wireless networks using ACK-DIV technique amp; FEC. In *2015 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6, 2015.

[Lau23] Lucas Laursen. No more "no service": Cellphones will increasingly text via satellite. *IEEE Spectrum*, 60(1):52–55, 2023.

[LD12] Lin Liu and Xiaoshe Dong. Evaluating Packet-level Forward Error Correction: 1-D interleaved parity codes. In *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, volume 1, pages 370–375, 2012.

[Li07] Adam H. Li. RTP Payload Format for Generic Forward Error Correction. RFC 5109, December 2007.

[LIB+17] Adam Langley, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Alistair Riddoch, Wan-teh Chang, Zhongyi Shi, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, and Ian Swett. The QUIC Transport Protocol. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*, pages 183–196, New York, New York, USA, 2017. ACM Press.

[LLC12] Ming Li, Andrey Lukyanenko, and Yong Cui. Network coding based multipath TCP. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 25–30. IEEE, 2012.

[LLO+16] Ming Li, Andrey Lukyanenko, Zhonghong Ou, Antti Ylä-Jääski, Sasu Tarkoma, Matthieu Coudron, and Stefano Secci. Multipath Transmission for the Internet: A Survey. *IEEE Communications Surveys & Tutorials*, 18(4):2887–2925, 2016.

[LLT+13] Ming Li, Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, and Antti Ylä-Jääski. Tolerating path heterogeneity in multipath TCP with bounded receive buffers. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 375–376. ACM, 2013.

[LLT+14] Ming Li, Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, and Antti Ylä-Jääski. Tolerating path heterogeneity in multipath TCP with bounded receive buffers. *Computer Networks*, 64:1–14, 2014.

[LM20] Yanmei Liu and Yunfei Ma. Mpquic use cases. `https://github.com/quicwg/wg-materials/blob/main/interim-20-10/MPQUIC%20use%20cases.pdf`, Oct 2020. Accessed: 25th April, 2023.

[LMC+22] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-00, Internet Engineering Task Force, 02 2022. Work in Progress.

[LMC+23] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-04, Internet Engineering Task Force, March 2023. Work in Progress.

[LMH+21] Yanmei Liu, Yunfei Ma, Christian Huitema, Qing An, and Zhenyu Li. Multipath Extension for QUIC, September 2021. Work in Progress.

[LMZG97] Hang Liu, Hairuo Ma, Magda El Zarki, and Sanjay Gupta. Error Control Schemes for Networks: An Overview. *Mob. Networks Appl.*, 2(2):167–182, 1997.

[LNTG17] Yeon-sup Lim, Erich M Nahum, Don Towsley, and Richard J Gibbens. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 147–159, 2017.

[LSR+18] E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. K. Ramakrishnan. Robustness of IoT Application Protocols to Network Impairments. In *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 97–103, 2018.

[MCM+22] Francois Michel, Alejandro Cohen, Derya Malak, Quentin De Coninck, Muriel Medard, and Olivier Bonaventure. FlEC: Enhancing QUIC With Application-Tailored Reliability Mechanisms. *IEEE/ACM Transactions on Networking*, pages 1–14, 2022.

[MDB19] Francois Michel, Quentin De Coninck, and Olivier Bonaventure. QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, may 2019.

[NXHS14] Dan Ni, Kaiping Xue, Peilin Hong, and Sean Shen. Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath tcp in lossy networks. In *2014 23rd international conference on computer communication and networks (ICCCN)*, pages 1–7. IEEE, 2014.

[Pau23] Rui Paulo. Implementations · quicwg/base-drafts Wiki. `https://github.com/quicwg/base-drafts/wiki/Implementations`, 2023. Accessed: 25th April, 2023.

[PKS22] Tommy Pauly, Eric Kinnear, and David Schinazi. An Unreliable Datagram Extension to QUIC. RFC 9221, March 2022.

[Pos80] Jon Postel. User Datagram Protocol. RFC 768, August 1980.

[Pos81a] J. Postel. Internet Protocol. RFC 791, September 1981.

[Pos81b] Jon Postel. Transmission Control Protocol. STD 793, RFC Editor, September 1981.

[RE16] Justin Ridgewell and Hala Elaarag. NCTCP: A Network Coded TCP Protocol. *Simulation Series*, 48(3):39–46, 2016.

[RGR+17] Alexander Raake, Marie-neige Garcia, Werner Robitza, Peter List, Steve Goring, and Bernhard Feiten. A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, volume 12, pages 1–6. IEEE, may 2017.

[RHB18] Alexander Rabitsch, Per Hurtig, and Anna Brunstrom. A stream-aware multipath quic scheduler for heterogeneous paths. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, EPIQ'18, page 29–35, New York, NY, USA, 2018. Association for Computing Machinery.

[RMQ20] Darijo Raca, Maelle Manifacier, and Jason J. Quinlan. goDASH — GO Accelerated HAS Framework for Rapid Prototyping. In *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–4. IEEE, 2020.

[RMSM20] Vincent Roca, François Michel, Ian Swett, and Marie-Jose Montpetit. Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC, March 2020. Work in Progress.

[RNB+18] John P. Rula, James Newman, Fabián E. Bustamante, Arash Molavi Kakhki, and David Choffnes. Mile high wifi: A first look at in-flight internet connectivity. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1449–1458, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[SBL+13] Golam Sarwar, Roksana Boreli, Emmanuel Lochin, Ahlem Mifdaoui, and Guillaume Smith. Mitigating receiver's buffer blocking by delay aware packet scheduling in multipath data transfer. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1119–1124, 2013.

[SCFJ03] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[SCQH19] Hang Shi, Yong Cui, Feng Qian, and Yuming Hu. Dtp: Deadline-aware transport protocol. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, APNet '19, page 1–7, New York, NY, USA, 2019. Association for Computing Machinery.

[SCW+18] Hang Shi, Yong Cui, Xin Wang, Yuming Hu, Minglong Dai, Fanzhao Wang, and Kai Zheng. {STMS}: Improving {MPTCP} throughput under heterogeneous networks. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 719–730, 2018.

[SKI19] Yurino Sato, Hiroyuki Koga, and Takeshi Ikenaga. TCP Using Adaptive FEC to Improve Throughput Performance in High-Latency Environments. *IEICE Transactions on Communications*, E102.B(3):537–544, 2019.

[SKK+08] Vicky Sharma, Shivkumar Kalyanaraman, Koushik Kar, KK Ramakrishnan, and Vijaynarayanan Subramanian. MPLOT: A transport protocol exploiting multipath diversity using erasure codes. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 121–125. IEEE, 2008.

[SKR06] Vijaynarayanan Subramanian, Shivkumar Kalyanaraman, and K.K. Ramakrishnan. An End-to-End Transport Protocol for Extreme Wireless Network Environments. In *MILCOM 2006 - 2006 IEEE Military Communications conference*, pages 1–7, 2006.

[SM23] Darius Saif and Ashraf Matrawy. An experimental investigation of tuning quic-based publish-subscribe architectures in iot, 2023.

[SMKK18] Tanya Shreedhar, Nitinder Mohan, Sanjit K Kaul, and Jussi Kangasharju. Qaware: A cross-layer approach to mptcp scheduling. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2018.

[SMRM20] Ian Swett, Marie-Jose Montpetit, Vincent Roca, and François Michel. Coding for QUIC, March 2020. Work in Progress.

[SPETG18] D. Stolpmann, C. Petersen, V. Eichhorn, and A. Timm-Giel. Extending On-the-fly Network Coding by Interleaving for Avionic Satellite Links. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, 2018.

[SR22] David Schinazi and Eric Rescorla. Compatible Version Negotiation for QUIC. Internet-Draft draft-ietf-quic-version-negotiation-14, Internet Engineering Task Force, December 2022. Work in Progress.

[SSM+11] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and João Barros. Network Coding Meets TCP: Theory and Implementation. *Proceedings of the IEEE*, 99(3):490–512, mar 2011.

[STkN22] Randall R. Stewart, Michael Tüxen, and karen Nielsen. Stream Control Transmission Protocol. RFC 9260, June 2022.

[Swe16] Ian Swett. QUIC FEC v1. `https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit`, 2016. Accessed: 23rd April, 2023.

[SWZ+20] Xiang Shi, Lin Wang, Fa Zhang, Biyu Zhou, and Zhiyong Liu. Pstream: Priority-based stream scheduling for heterogeneous paths in multipath-quic. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8, 2020.

[SWZL19] Xiang Shi, Lin Wang, Fa Zhang, and Zhiyong Liu. Fstream: Flexible stream scheduling and prioritizing in multipath-quic. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 921–924, 2019.

[TFH+07] Tomoaki Tsugawa, Norihito Fujita, Takayuki Hama, Hideyuki Shimonishi, and Tutomu Murase. TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee. In *Packet Video 2007*, pages 294–301, 2007.

[Tho21] Martin Thomson. Version-Independent Properties of QUIC. RFC 8999, May 2021.

[TOHI17] Fumiya TESHIMA, Hiroyasu OBATA, Ryo HAMAMOTO, and Kenji ISHIDA. TCP-TFEC: TCP Congestion Control based on Redundancy Setting Method for FEC over Wireless LAN. *IEICE Transactions on Information and Systems*, E100.D(12):2818–2827, 2017.

[TSR19] P. Truchly, M. Sith, and R. Repka. End-to-end Packet Loss Differentiation Algorithms and Their Performance in Heterogeneous Networks. In *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 777–783. IEEE, nov 2019.

[TT21] Martin Thomson and Sean Turner. Using TLS to Secure QUIC. RFC 9001, May 2021.

[VFR+18] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath QUIC: A Deployable Multipath Transport Protocol. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, may 2018.

[VW21] Vu Anh Vu and Jan Wolff. Supporting Delay-Sensitive Applications with Multipath QUIC and Forward Erasure Correction. In *Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 95–103, New York, NY, USA, nov 2021. ACM.

[WAB+20] Hongjia Wu, Özgü Alay, Anna Brunstrom, Simone Ferlin, and Giuseppe Caso. Peekaboo: Learning-based multipath scheduling for dynamic heterogeneous environments. *IEEE Journal on Selected Areas in Communications*, 38(10):2295–2310, 2020.

[WBRP18] Peng Wang, Carmine Bianco, Janne Riihijärvi, and Marina Petrova. Implementation and Performance Evaluation of the QUIC Protocol in Linux Kernel. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '18, page 227–234, New York, NY, USA, 2018. Association for Computing Machinery.

[WCF+21] Hongjia Wu, Giuseppe Caso, Simone Ferlin, Özgü Alay, and Anna Brunstrom. Multipath scheduling for 5g networks: Evaluation and outlook. *IEEE Communications Magazine*, 59(4):44–50, 2021.

[WCWC17] Jiyan Wu, Bo Cheng, Ming Wang, and Junliang Chen. Priority-Aware FEC Coding for High-Definition Mobile Video Delivery Using TCP. *IEEE Transactions on Mobile Computing*, 16(4):1090–1106, 2017.

[WGP+17] Simon Wunderlich, Frank Gabriel, Sreekrishna Pandi, Frank H.P. Fitzek, and Martin Reisslein. Caterpillar RLNC (CRLNC): A Practical Finite Sliding Window RLNC Approach. *IEEE Access*, 5:20183–20197, 2017.

[WGX19] Jing Wang, Yunfeng Gao, and Chenren Xu. A multipath quic scheduler for mobile http/2. In *Proceedings of the 3rd Asia-Pacific Workshop on Networking 2019*, APNet '19, page 43–49, New York, NY, USA, 2019. Association for Computing Machinery.

[WL06] Mea Wang and Baochun Li. How Practical is Network Coding? In *200614th IEEE International Workshop on Quality of Service*, pages 274–278. IEEE, jun 2006.

[WYC+15] Jiyan Wu, Chau Yuen, Bo Cheng, Ming Wang, and Junliang Chen. Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing*, 15(9):2345–2361, 2015.

[WYC+16] Jiyan Wu, Chau Yuen, Bo Cheng, Yuan Yang, Ming Wang, and Junliang Chen. Bandwidth-efficient multipath transport protocol for quality-guaranteed real-time video over heterogeneous wireless networks. *IEEE Transactions on Communications*, 64(6):2477–2493, 2016.

[YWA14] Fan Yang, Qi Wang, and Paul D. Amer. Out-of-order transmission for in-order arrival scheduling for multipath tcp. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 749–752, 2014.

[ZGL+19] Han Zhang, Haijun Geng, Yahui Li, Xia Yin, Xingang Shi, Zhiliang Wang, Qianhong Wu, and Jianwei Liu. DA&FD–Deadline-Aware and Flow Duration-Based Rate Control for Mixed Flows in DCNs. *IEEE/ACM Transactions on Networking*, 27(6):2458–2471, 2019.

[ZML+21] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, Qing An, Hai Hong, Hongqiang Harry Liu, and Ming Zhang. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 418–432, New York, NY, USA, 2021. Association for Computing Machinery.