IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 1, 2021, accepted September 28, 2021, date of publication October 5, 2021, date of current version October 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3118112

Robust QUIC: Integrating Practical Coding in a Low Latency Transport Protocol

MIHAIL ZVEREV[1], PABLO GARRIDO[2], FÁTIMA FERNÁNDEZ[1], JOSU BILBAO[1], (Senior Member, IEEE), ÖZGÜ ALAY[3,4], (Member, IEEE), SIMONE FERLIN[5], ANNA BRUNSTROM[6], (Member, IEEE), AND RAMÓN AGÜERO[7], (Senior Member, IEEE)

[1]Information and Communication Technologies Area, Ikerlan Technology Research Centre, Basque Research Technology Alliance (BRTA), 20500 Arrasate/Mondragón, Spain
[2]Nemergent Solutions, 48950 Erandio, Spain
[3]Department of Informatics, University of Oslo, 0373 Oslo, Norway
[4]Department of Mobile Systems and Analytics, Simula Metropolitan Center for Digital Engineering, 0167 Oslo, Norway
[5]Ericsson AB, 164 40 Kista, Sweden
[6]Department of Mathematics and Computer Science, Karlstad University, 651 88 Karlstad, Sweden
[7]Department of Communication Engineering, University of Cantabria, 39005 Santander, Spain

Corresponding author: Mihail Zverev (mzverev@ikerlan.es)

This work was supported in part by the Basque Government through the Elkartek Program under the Hodei-x Project under Agreement KK-2021/00049; in part by the Spanish Government through the Ministerio de Economía y Competitividad, Fondo Europeo de Desarrollo Regional (FEDER) through the Future Internet Enabled Resilient smart CitiEs (FIERCE) under Grant RTI2018-093475-AI00; and in part by the Industrial Doctorates Program of the University of Cantabria under Grant Call 2019.

ABSTRACT We introduce rQUIC, an integration of the QUIC protocol and a coding module. rQUIC has been designed to feature different coding/decoding schemes and is implemented in go language. We conducted an extensive measurement campaign to provide a thorough characterization of the proposed solution. We compared the performance of rQUIC with that of the original QUIC protocol for different underlying network conditions as well as different traffic patterns. Our results show that rQUIC not only yields a relevant performance gain (shorter delays), especially when network conditions worsen, but also ensures a more predictable behavior. For bulk transfer (long flows), the delay reduction almost reached 70% when the frame error rate was 5%, while under similar conditions, the gain for short flows (web navigation) was $\approx$ 55%. In the case of video streaming, the QoE gain (p1203 metric) was, approximately, 50%.

INDEX TERMS QUIC, transport layer, FEC, network coding, performance assessment.

I. INTRODUCTION (MOTIVATION)

QUIC has recently emerged as an alternative to widespread Transmission Control Protocol (TCP). Originally fostered by Google [1], it has several advantages that make it an appealing choice, especially for specific scenarios. The main advantages of QUIC over TCP are: (1) reduction of the connection-establishment latency, (2) avoidance of head-of-line blocking, and (3) ease of updating processes. Its use as a transport protocol alternative has been steadily increasing in the latest years. At the time of writing (August 2021), approximately 5.7% of all websites are using it [2]. This increasing interest has been further strengthened by standardization efforts [3]. Key players such as Google, Apple, Microsoft, and Facebook have developed their own QUIC implementations [4], which should further instigate its widespread adoption.

QUIC enables the integration of additional features and modules that may improve its performance under certain scenarios. In particular, the integration of coding techniques, such as Forward Erasure/Error Correction (FEC), might bring important benefits, for example, for short flows, where it would enable the recovery of tail losses. In addition, it could also be beneficial for services with (almost) real-time requirements, since it would allow the recovery of packet losses regardless of the underlying Round Trip Time (RTT).

Some studies have integrated coding techniques within QUIC, with the main goal of enhancing its behavior over packet-erasure channels [5]–[9]. Although some of the preliminary results are promising, there is not yet a clear characterization of the benefits that the use of coding techniques might bring, when using QUIC over different wireless links.

The associate editor coordinating the review of this manuscript and approving it for publication was Arun Prakash.

In this paper we discuss the integration of a coding module within a real QUIC implementation (in go language). We carry out an in-depth characterization of the performance of this module over various wireless technologies (with different delays, bandwidths, and packet error rate characteristics). We discuss the implementation choices for the coding module and describe how it is integrated within QUIC. Finally, we use the ns-3 simulator to assess the performance of the proposed scheme. We compare the behavior of our solution with the original QUIC protocol considering long flows (bulk transfer), short flows (web page download), and real-time services (video streaming). The results show that the integration of the coding module yields a significant performance enhancement for all considered traffic patterns and technologies. To the best of our knowledge, this is the most complete and thorough assessment of the combination of QUIC with a coding functionality, and the first one to consider realistic real-time traffic patterns, such as video streaming and the DASH protocol. Further, we provide our implementation as an open-source code in a GitHub repository.[1]

We structure this paper as follows: Section II positions our paper with regards to related work and provides some background of the QUIC transport protocol and coding approaches applied to transport layer. Section III discusses our implementation architecture, its integration with the QUIC protocol, and its main components and algorithms. Section IV describes our FEC implementation, including its operation on the encoder and decoder sides. Section V provides further implementation details, such as packet structures and some of the assumptions that were made. Finally, Section VI discusses our experimental setup and evaluation results. In Section VII, we discuss how we plan to broaden rQUIC implementation. Finally, we conclude the paper in Section VIII.

II. BACKGROUND AND RELATED WORK

We start by depicting the main characteristics of the QUIC transport protocol, and then discuss how it might benefit from the integration of a coding module. Finally, we discuss some of the most relevant related works that proposed a combination of a transport protocol and a coding module.

A. QUIC

QUIC is a connection-oriented transport protocol that follows the traditional client/server architecture. QUIC was initially proposed by Google [1] and was recently standardized by the Internet Engineering Task Force (IETF) [3]. While retaining some of the key TCP features, such as reliable delivery and congestion control, QUIC is built on top of User Datagram Protocol (UDP), with the aim of solving the challenges associated with protocols running on top of TCP, such as connection-setup time, head-of-line blocking, and middlebox interference.

[1] https://github.com/pgOrtiz90/quic-go-fec

QUIC establishes a secure connection and reduces connection-setup latency by including TLS 1.3, allowing application data to be sent during the very first round-trip time of a connection (zero-RTT approach). To avoid delays produced by head-of-line blocking, the information of a QUIC connection is arranged in streams. The streams that have data in a lost packet are the only ones that are blocked waiting for the required retransmission, while the rest are still being sent.

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 the identifiers are not repeated if a loss is detected. 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 acknowledgment is received. As a result, QUIC achieves more accurate RTT measurements and can to identify spurious retransmissions [10].

QUIC loss detection algorithms are acknowledgement-based, with a probe timeout to ensure that the acknowledgements are received. These algorithms are activated according to the following specific cases:

- 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 packet was sent, and the subsequent one was not acknowledged.

Middleboxes are networking elements that manage TCP traffic, analyze TCP segments and modify them with the aim of leveraging an optimal performance [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 [1]. As for middleboxes meddling the connection, QUIC encrypts its packets, leaving only unprotected header fields that are required to identify a session at an endpoint [12]. In addition to security reasons, this encryption is used to avoid any interaction with middleboxes, and thus, protocol ossification [1]. 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 [13], 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 [14]. 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 [7].

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

B. FEC AND NETWORK CODING

Traditionally, there are two main causes of information loss events during communication: congestion events and transmission errors. Reliable protocols, such as TCP [15] and QUIC [10], recover lost information using Automatic Repeat reQuest (ARQ) mechanisms. These consist of signaling lost information and taking care of its retransmission. On the other hand, FEC [16] techniques promote a different approach, allowing the recovery of lost chunks of information by sending additional packets, built as combinations of the original data. 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.

Ahlswede *et al.* introduced the concept of Network Coding (NC), where intermediate nodes can perform coding operations on ongoing communication flows [17]. Later, Katabi *et al.* proposed in [18] *intra-flow* NC, a particular flavor of NC, to enhance end-to-end communications. In summary, it can be described as an FEC operating over each hop on the communication path (between the two endpoints), with NC-enabled intermediate nodes.

In addition, Ho *et al.* proposed in [19], [20] Random Linear Network Coding (RLNC), a coding technique where all the transmissions correspond to a random combination of the original data chunks, known as source symbols. These are grouped into generations (size N) and combined to create coded symbols. Whenever the destination (or an intermediate node) receives 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 and Li have shown that the use of sparse coding matrices, where some of the coding coefficients are zero, can alleviate this problem [21]. 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 [22]. 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. One modification is to overlap the generations, a coding approach known as convolutional coding [23]. A systematic RLNC with overlapped generations was discussed in [24] and further extended with ARQ in [25]. 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 was exploited in [26], focusing on concurrent generations rather than on a coding window.

A convolutional coding scheme that overlaps generations yields an overall latency reduction [24]. 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 [27], 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, consecutive 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 [28]. 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.

C. CODING VARIANTS AND CHARACTERISTICS

This section covers coding features that have been particularly considered for this work, introducing naming

conventions we will use throughout the paper, since the terminology used in rQUIC development slightly differs from the one recommended in [29].

One such difference is the use of the *generation* concept. Coding operations were originally intended for RLNC operation. Hence, we widely use the term *generation*, a group of source symbols used to build coded symbols. However, as will be explained later, the rQUIC design that we discuss here aims at systematic convolutional coding schemes. A more appropriate term in this context is *the encoding window* [29].

Furthermore, the *code rate* is defined, from an implementation point of view, as in [6]: the rate between source and coded symbols, which is different from the definition in [29]. 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. To reduce the number of cases where FEC would not be able to help, we assume that the entire packet corresponds to one symbol. Hereinafter, 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 II-B, 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.

As introduced in Section II-A, 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. Our 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, we adjust the number of redundancies 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 [28] 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 [30] 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:

$$Q = \frac{g}{r\varphi} \tag{1}$$

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. Fig. 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 sources 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 our overlap definition, the last overlapping generations also need to be shortened. In the example illustrated in Fig. 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 communication endpoint. Fig. 1 shows a short transmission of only 15 source packets with losses. Lost packets are marked with a zig-zag strike-through. Until

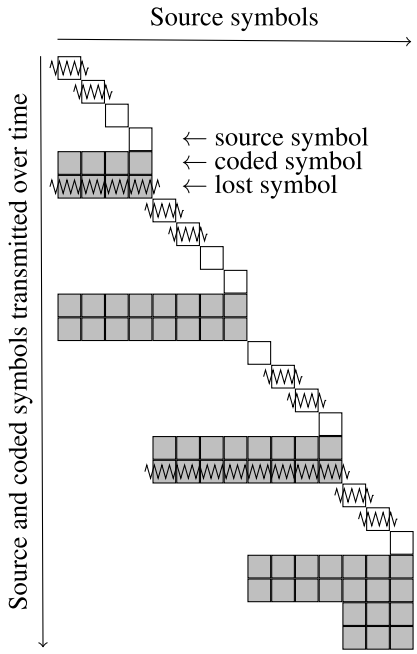


FIGURE 1. Graphical representation of convolutional coding for 15 source symbols protected with 2 overlapping generations of 8 symbols and 2 coded symbols per generation.

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.

D. CODING AT TRANSPORT LAYER

Various studies 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, we review some of the most important examples.

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 [31], 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 [32], which is an extension of TCP with an adaptive coding scheme to improve video streaming. TCP-AFEC hides the recovered losses from TCP congestion control. 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 [33] 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 [34]. They interleave generations, as was also suggested by [28], 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 congestion control mechanisms, allowing high goodput to be maintained. Congestion avoidance relies on Explicit Congestion Notification (ECN) signals.

When coding techniques are applied to reliable transport protocols with congestion control, 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 [35]. When coding techniques are applied to a reliable transport protocol's data flow, that is, above the transport layer as in [31], coded symbols and packet losses are visible to the congestion control. 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 [36] some of the most relevant CCAs that have been proposed.

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) [37] over UDP. RTP specifies different FEC extensions, such as the generic FEC specification [38] and XOR coding with interleaving [39], the latter being extensively evaluated in [40]. Another example of UDP video streams protected with an FEC is [41]. 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 [42]. 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 [8] focuses on how to implement a generic FEC scheme, while [9] focuses specifically on RLC. Furthermore, Michel *et al.* presented an FEC extension with a fixed code rate, assessing its performance over constrained links [5]. This extension is subsequently converted into a portable plugin, integrated along with multipath and other transport features in the *pluginized* QUIC [7]. Garrido *et al.* presented the first version of rQUIC [6], an extension of QUIC with an adaptive FEC. We improve the initial implementation by designing a more generic scheme, which might be configured to feature different coding solutions. As discussed in Section VI, the results demonstrate that this implementation yields a better performance than the original one. In addition, we also assess the behavior that rQUIC exhibits when used with services having real-time requirements, 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. The authors of [43] 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 [43]–[46]. We argue that one of the main obstacles lies in QUIC philosophy, since it prevents a tight interaction with middleboxes. If no interaction with middleboxes is used, 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.

The benefits offered by coding techniques are not limited to improving the communication reliability. Multipath (MP) is a feature of transport protocols that enables data exchange between two endpoints, either over different network interfaces or different IP addresses. Selecting the right path to transmit a new packet is not a trivial task, and it can be seen as an entire research line on its own [47]. However, coding techniques can greatly simplify this task, as in MPLOT [48]. This transport protocol uses the aggregated loss rate to build redundancies and distribute the resulting generation with its coded packets across all paths. MPLOT acknowledges the generations instead of packets. In this way, source and coded packets belonging to the same generation can be transmitted over any path and in any order, as long as each path's transmission window is respected. Another approach to combine coding and MP could be sending source and coded packets on different paths, a strategy followed in fountain code-based MPTCP (FMTCP) to mitigate the negative impact of path heterogeneity [49]. Furthermore, the combination of coding and multiple path networking can be efficiently handled from the application layer, as shown in [41]. At the time of writing, there are multiple approaches and implementations of multipath extension for QUIC [50]–[53], including additional extensions, such as a FEC module [7]. However, there is still no clear evaluation of the potential benefits of the simultaneous use of multipath and coding techniques. The rQUIC design considers the main multipath functionalities, so that both schemes can be combined. In this paper we do not assess the potential benefits of this integration, but will be addressed in our future work.

III. ARCHITECTURE OVERVIEW

We consider traditional client/server communication, where the receiver (client) downloads the information sent by the transmitter (server). We also assume that there may be circumstances (congestion events or packet erasures) where losses occur, and not all transmitted information correctly arrives at the receiver. We propose the use of a coding scheme to improve the original recovery mechanisms included in QUIC, which are based on an ARQ scheme.

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. Fig. 2 illustrates the architecture of the proposed extension. The main components are the encoder, integrated within the transmitter operation, and the decoder, 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.

A. 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 IV.

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. We introduce three main rQUIC packet types, whose signaling is discussed in more detail in Section V-A.

- 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.

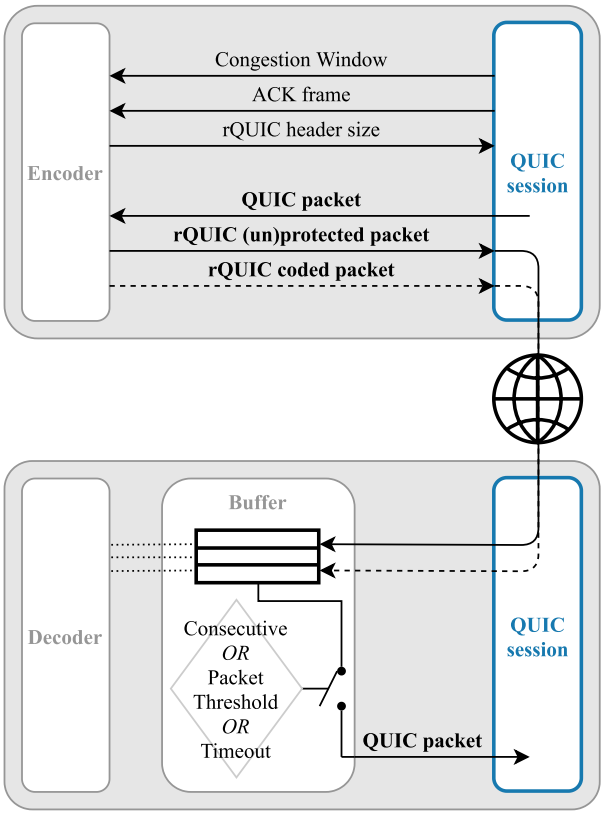


FIGURE 2. rQUIC architecture.

B. 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 the 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.

1) RECEIVER BUFFER

As mentioned previously, the decoder buffers the packets for further decoding operations. In addition, this would also allow the FEC module to recover packets that might have been lost. Once source packets are passed to the QUIC session, the latter might send an ACK frame[2] 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 the 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[3] from a newer generation were received. At this point we assume 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 the 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 [14], in which an ACK must be sent for any packet. Therefore, we define BTO as the maximum ACK delay D with a certain margin M:

$$BTO := D - M \tag{2}$$

We set the margin M to 1 ms. The default maximum ACK delay is 25 ms [14]. In the `quic-go` implementation, the endpoints set their ACK delays to 26 ms, and thus BTO in our experiments is set to 25 ms.

IV. 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.

A. 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 we are using in the experiments reported in this paper, both XOR and systematic RLC are supported.

The distribution and frequency of CPs depend on the following parameters, which are introduced in Section II-C:

- *Code Rate (Q)*: the rate between source and coded packets.

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

[3]3 is the recommended value for `kPacketThreshold`, packet reordering threshold [10].

- *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.

The rQUIC encoder can adjust the number of CPs it sends based on network conditions (more details are given in Section IV-B). 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 Eq. 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:

$$gen.\,complete := (g + 1 > Qr\varphi)\ or\ (g \geq g_{max}) \tag{3}$$

Redundancies per generation (r) and overlap (φ) are static parameters that are set at the beginning of the connection. As explained in Section II-C, 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 $\varphi = 1$.

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. References [12] 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.

B. DYNAMIC CODE RATE

To correct packet losses in environments with unpredictable and varying loss rates, our extension includes the dynamic code rate. The adaptive approach helps recover losses without overloading the link with the CPs. We use the algorithm proposed in [6], [54], [55].

The code rate Q changes depending on the Residual Loss (RL), defined as the rate between lost and delivered packets. The 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 our implementation, $\gamma = 0.01$ and $\delta = 0.33$.

Algorithm 1 Adaptive Code Rate in rQUIC

```
Q ← Q_init
if RL > γ then
    Q ← Q × (1 − δ)
else
    Q ← Q × (1 + δ)
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, we keep the code rate below the CWND (expressed in terms of packets). To convert the CWND from bytes to packets, we use the maximum QUIC packet size allowed by the implementation.

V. IMPLEMENTATION DETAILS

Hereafter, we will refer to the QUIC specification corresponding to the draft version 29 [14], which was implemented in the base code we chose at the time of experimentation.

A. 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 II. Although extending QUIC with NC is beyond the scope of this study, we decided to leave coding coefficients unencrypted, so that we could experiment with it in the future. 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.

Fig. 3 illustrates the new fields introduced by rQUIC, which are explained in more detail below.

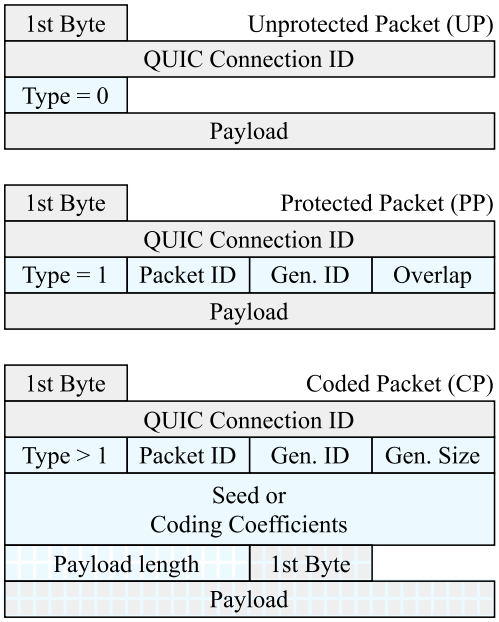


FIGURE 3. New fields for different types of rQUIC packets.

1) HEADERS

The fixed part of the short header packets ends with the Destination Connection ID field [56], 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. We used 1 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 II-C, PPs can belong to multiple generations; therefore, they indicate the most recent generation to which they belong.

Owing to the packet ID field, we know 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 [6] (cf. Fig. 2 in that paper), we set the maximum generation size 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. Our implementation 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.

2) PAYLOADS

QUIC leaves unprotected the Connection ID field, as well as some bits of the first byte [14]. 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 Fig. 3.

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.

B. 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 V-A1, 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. These comparisons might become more complex with large generation sizes and a large number of overlapped generations. Since in this work we evaluate the performance with an XOR scheme without overlapping, packet ID comparison remains quite simple. Nevertheless, we included in our implementation algorithms for the careful detection of obsolete packets. We do not provide more details here for the sake of text simplicity.[4]

C. 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 [57], they can be completely avoided by applying coding techniques before encryption, leaving coefficients encrypted, as in [5]. 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. Dealing with privacy issues is beyond the scope of this paper, but we are aware of the next steps to address this issue. Our implementation 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 V-A2, 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 congestion control algorithms 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 [58]. On the other hand, [35] suggests that FEC coding could likely benefit communications with persistent non-congestion losses, which is often the case for wireless networks.

Hence, our main assumption is that isolated losses are mostly caused by transmission errors rather than actual congestion. Our design also aims to avoid tampering with the operation of congestion control algorithms. In this sense, rQUIC is configured to send only one CP per generation, and the generation sizes are limited to the congestion window. We argue that these measures do not hinder the performance of real congestion events. In any case, this work focuses on the QUIC/FEC interaction, while we leave a more in-depth analysis of the complex interactions between FEC and congestion control for our future work.

VI. EVALUATION RESULTS

In this section we discuss the evaluation that we carried out to assess the performance yielded by rQUIC and compare its behavior with that exhibited by the traditional QUIC operation.

[4]The reader might check these in the corresponding implementation

A. SETUP

To carry out the experiments, we exploited the ns-3 simulator[5] version 3.30.1. All networks have the same topology, as depicted in Fig. 4. The corresponding binary files of rQUIC test applications are placed in lxc containers[6] (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.

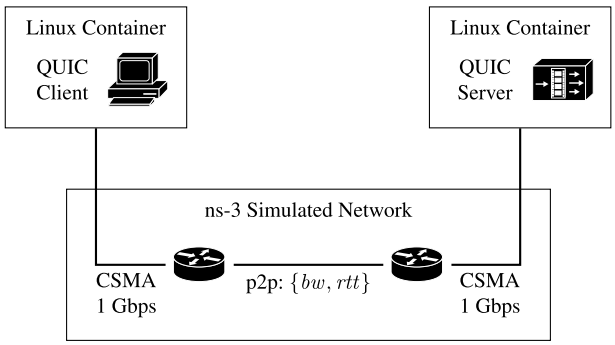


FIGURE 4. Emulation scenario.

The point-to-point link behavior depends on three parameters: Bandwidth (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 [6]. 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 IV. The residual loss measurement period is of 3 RTTs, and it is averaged over 3 measurement periods. As explained in Section IV-A, 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.

B. 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.

[5]https://www.nsnam.org/
[6]https://linuxcontainers.org/

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 Eq. 4. 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.

$$\xi = \frac{\text{rQUIC Completion Time}}{\text{QUIC Completion Time}} \tag{4}$$

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:

$$\hat{O} = \frac{\text{CP}}{\text{PP} + \text{CP}} \tag{5}$$

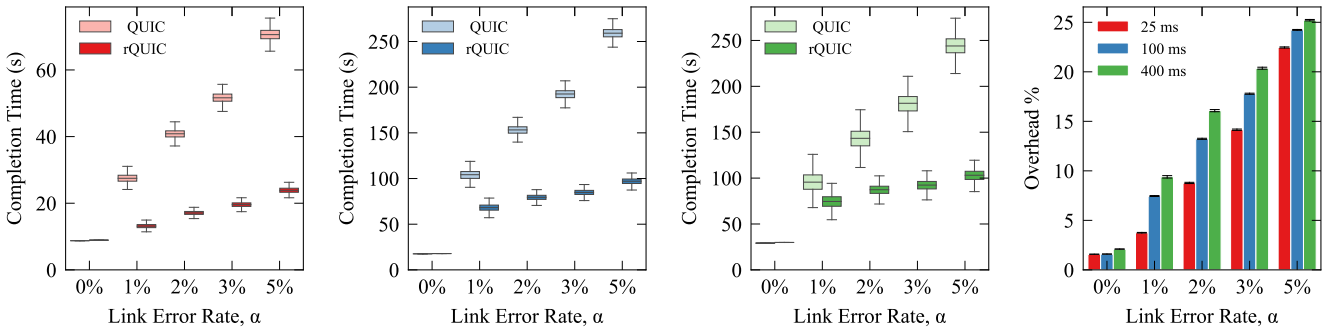
Fig. 5 shows the results of the bulk transmission experiment. We ran more than 1150 independent experiments for each configuration. Whisker plots were used to represent the overall delay observed for both QUIC and rQUIC. The boxes represent the interquartile range with the median mark inside of the boxes. The whiskers represent Tukey fences. The overhead bar plots (Fig. 5d) 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 (Fig. 5a), 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. Fig. 5d shows that at a 0% loss 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 5a–5c.

Fig. 5d 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



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

FIGURE 5. Completion time for bulk transfer and the overhead generated by coded packets.

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 III, 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 VI-E.

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 Fig. 5d.

The overhead $\hat{O}$ defined in Eq. 5 can be rewritten as follows, where $\bar{g}$ is the average generation size, and $\bar{Q}$ is the average code rate.

$$\hat{O} = \frac{r\varphi}{\bar{g} + r\varphi} = \frac{1}{\frac{\bar{g}}{r\varphi} + 1} = \frac{1}{\bar{Q} + 1} \tag{6}$$

From Eq. 6 the average code rate can be calculated as:

$$\bar{Q} = \frac{1}{\hat{O}} - 1 \tag{7}$$

Fig. 5d shows that at a loss rate of 5%, the overhead is above 20%, surpassing 25% for the satellite network. Applying Eq. 7, 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 [10]. 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.

Fig. 6 compares the results with those obtained in [6] 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 [5] because they did not assess the performance of their solution with bulk transfers.

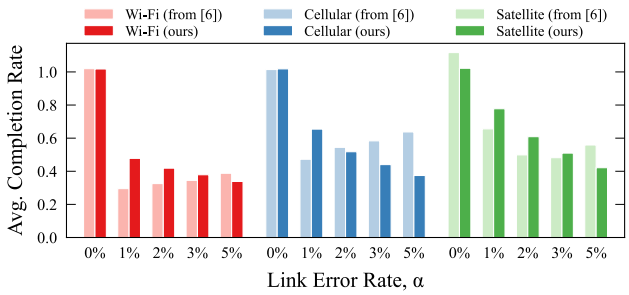
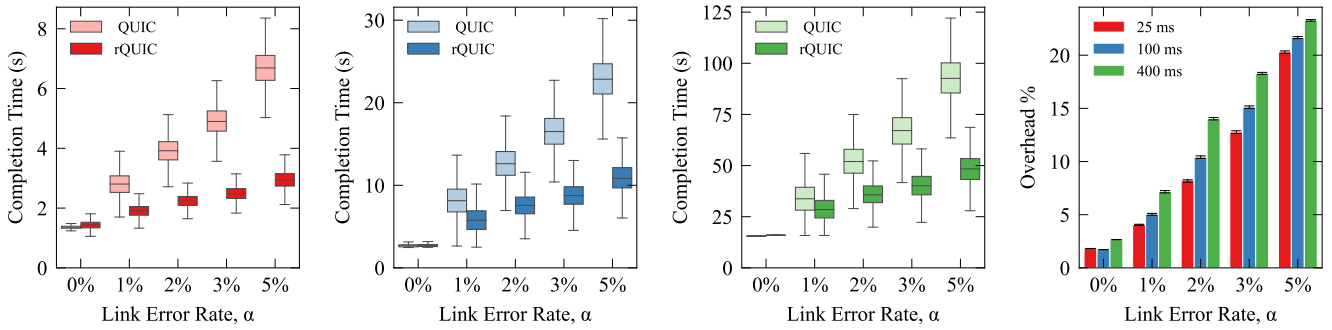


FIGURE 6. Bulk transfer average completion rates for the current rQUIC and its previous version presented in [6].

C. 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



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

FIGURE 7. **Completion time for web page download and the overhead generated by coded packets.**

emulate web traffic, we used the tool Epload,[7] which saves downloaded objects 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.[8]

Fig. 7 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 Fig. 7a) is that QUIC completion times have much greater dispersion than rQUIC. This aspect might be 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, Fig. 8 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.

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

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

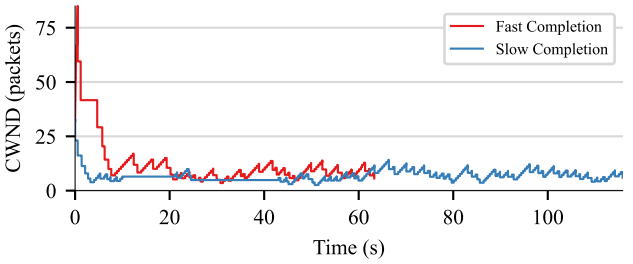


FIGURE 8. **Original QUIC CWND evolution for a fast and a slow web page download over the satellite link at 5% loss.**

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 Fig. 7d 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, Fig. 7d shows that, as the RTT increases, the overhead increases, although, as can be seen in Figures 7a–7c, 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 VI-E.

As was done for the bulk experiment, Fig 9 compares the performance exhibited by the rQUIC implementation discussed here with the one discussed in [6]. 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

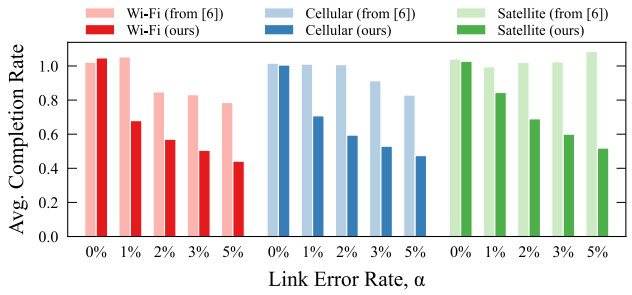


FIGURE 9. Web page download average completion rates for the current rQUIC and its previous version presented in [6].

better than that obtained in [6]. 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 [6].

The results presented by Michel *et al.* in QUIC-FEC [5] 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 [5] imply the transmission of short files, and they are thus not comparable with our results. While QUIC-FEC [5] 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.

D. 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 HTTP (DASH) is a standardized technique [59] for sending video streams over HTTP. We used `go-dash`,[9] an implementation in the GO language of the DASH protocol, published in [60], as well as a DASH testbed.[10]

[9]https://github.com/uccmisl/godash
[10]https://github.com/uccmisl/godashbed

The DASH server streams "Tears of Steel" short movie. We used the p1203 [61] 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.

Fig. 10 depicts the average p1203 score with 95% confidence intervals for DASH video streaming. For the Wi-Fi network (Fig. 10a), 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 Fig. 10c. 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.

To complement the previous p1203 score results, Fig. 11 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 Fig. 11, the results obtained for Wi-Fi and cellular technologies.

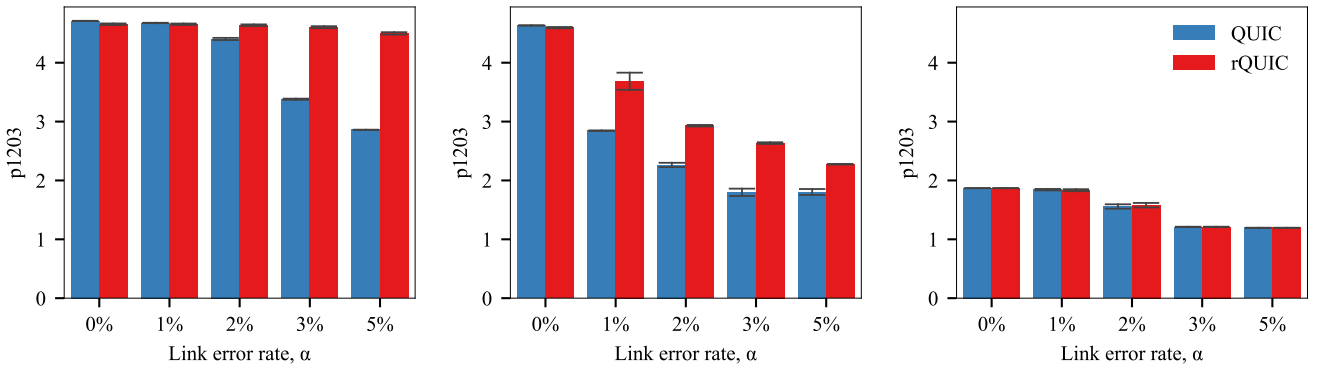
Fig. 11a shows that when the loss rate is $\geq 2\%$, rQUIC is able to transmit higher resolution frames than QUIC. Even for the worst conditions (5% loss rate), rQUIC can maintain most of the resolutions at 1280×720, while with QUIC most of the time the transmitted resolution is 640×360. For better 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 (Fig. 11b). 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.

E. BUFFER TIMEOUT EXPLORATION

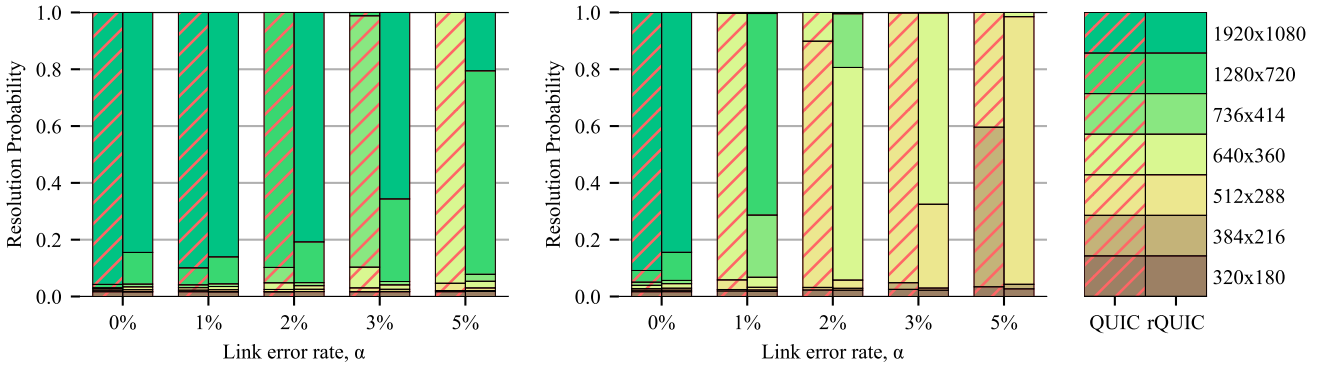
We observed that the FEC performance worsened for larger RTT values. Our initial assumption is that this degradation is caused by an inappropriate BTO value. However, using a fixed BTO may not be a good solution for changing network conditions.

If packet transmission rate is paced, the optimum BTO can be searched according to the pacing algorithm. Based



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

FIGURE 10. QUIC and rQUIC performance comparison using p1203 score.



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

FIGURE 11. QUIC and rQUIC performance comparison in terms of achieved video resolutions.

on the recommendations in [10], `quic-go` implements its sending rate as shown in Eq. 8, 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.

$$sending\ rate = \frac{5}{4}\frac{\text{CWND}}{\text{sRTT}} \tag{8}$$

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 12 and 13 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).

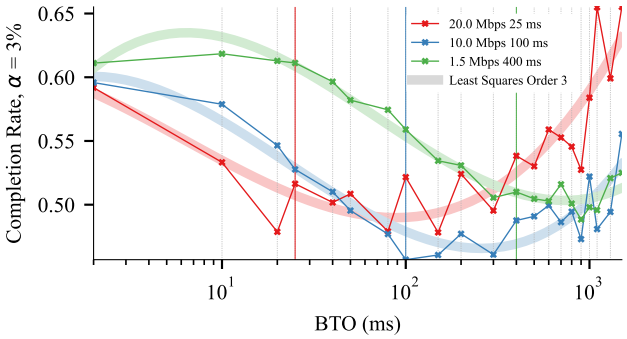


FIGURE 12. Average completion rates for different values of BTO in web page download.

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 12 and 13 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

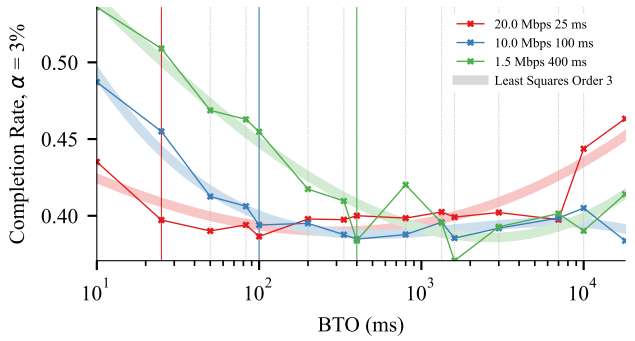


FIGURE 13. Average completion rates for different values of BTO in bulk transfer scenario.

and the corresponding RTT for the web traffic experiments. In this sense, the minimum values in Fig. 12 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 Fig. 13 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 12 and 13 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 5 and 7) are greater than the optimum BTO, and completion rates are growing (see Figures 13 and 12). This growth is more difficult to appreciate in Fig. 13 for cellular and satellite links, because the communication is longer, and the observed growth rate is still low.

VII. DISCUSSION AND FUTURE EXTENSIONS

In this work, we have seen that QUIC performance can be significantly improved using the adaptive FEC extension that we have integrated. However, more enhancements may be added to boost communication robustness and efficiency.

When CPs are inserted, they do not interact with the congestion control mechanism. This is sensible because the transmission of just 1 CP per generation should not have a strong impact on congestion control. However, we have seen, for both bulk transfer and short web page download experiments, that when loss rates grow, the number of CPs should not be neglected from the perspective of congestion control mechanisms. More specifically, at a 5% loss rate, the overhead in both experiments is above 20%, meaning that 1 CP is inserted every 4 or even 3 PPs. Therefore, it is important to establish an appropriate interaction between the FEC module and the congestion control scheme, which might comprise accounting CPs and the corresponding modifications to the adaptive code rate, introducing features such as the eventual skipping of CP transmission during congestion events. The latter would also require distinguishing between congestion and network losses.

In the FEC implementation presented in this work, all coding operations are performed over already encrypted packets to enable further research on QUIC improvements with NC. However, given that encryption is the last step in QUIC packet construction, performing coding operations on unencrypted payloads would greatly simplify the interaction between FEC and other features or extensions. For instance, this would greatly simplify the interaction with the congestion control scheme, leading to a congestion-aware FEC implementation, as the one introduced in [5]. The interaction between the coding module and the eventual multipath extensions can also be simplified. Secondary paths could be probed with CPs, which can be naturally acknowledged by the QUIC session, if coding techniques are applied before encryption. Pre-encryption FEC should not replace the existing solution because both can be combined to implement BATS coding techniques [62].

The FEC encoder presented in this work buffers only coded packets under construction, adding the original packets on the fly. Another possible approach would be to buffer the original (source) packets and build the coded packets immediately before their transmission. Although this would require more memory at the encoder, it would allow the selection of only unacknowledged source packets for building the coded ones. Another benefit would be the reduction of the decoder's buffer length because fewer source packets would be required to decode a coded one. Either of the two approaches should be selected, based on the constraints of the particular devices (transmitter and receiver), especially in terms of memory.

At the end of a communication, it is quite likely that the last generation could not be completed, since the number of the last PPs might be smaller than the generation size, and the corresponding CPs had not been transmitted. The losses of such packets, including tail losses of the very last ones, are recovered with QUIC retransmissions. To send protection packets for small generations, the encoder requires a rather short timeout for CP transmission. This would be particularly beneficial for short communications, where multiple packets are sent, but they might not be enough to complete a single generation (32 packets, as imposed by `quic-go` initial CWND), which could be the case for monitoring devices sending short reports every minute.

The decoder presented in this work delays PP delivery to allow the FEC to recover lost packets before the latter are retransmitted. We define a maximum delay time, the Buffer Timeout (BTO). To comply with the QUIC specification, BTO is defined as a function of `max_ack_delay`,

a transport parameter that is set at the beginning of the communication and never changed. The BTO exploration experiment (Section VI-E) revealed a significant boost in rQUIC performance when BTO was slightly greater than RTT. In future rQUIC updates, in order to define a more suitable BTO, the endpoints might estimate their RTTs during connection establishment or further extend the original QUIC functionality to renegotiate the `max_ack_delay` parameter after the connection has already been established, or upon detecting a significant change in the observed RTT. A less universal yet simpler solution would be to set the appropriate `max_ack_delay` from the application before the actual connection establishment.

The current implementation of the decoder prevents the delivery of PPs received after a loss was detected. The reason for this is to delay the corresponding ACK transmission, which triggers lost packet retransmission. This is not the safest way to delay the generation of ACK. At the time of writing, there are two active IETF drafts addressing the ACK frequency variation [63], [64]. We argue that an extension delaying ACK frame generation would allow the FEC module to recover lost packets, and thus avoid retransmissions, simplifying the decoder implementation. Another potential improvement in the decoder implementation is the modification of the QUIC behavior when an ACK is received, defining a new logic for delaying the retransmission of lost frames.

In this work we have used XOR coding for two reasons: (1) it corresponds to a scheme whose redundancies per generation r and generation overlap φ equal to 1, minimizing the generation size (and so the recovery latency) and the decoder's buffer length; (2) we have assumed that single loss events are due to wireless link impairments, and are not a consequence of congestion situations. However, wireless losses can also occur in bursts. Cardoso and de Rezende analyze in [65] Wi-Fi bursty losses and proposed a model for them. In their setup, they observed bursts that might have been rather long. The FEC module has been designed with the goal of correcting persistent non-congestion losses [35], which can already be achieved with the adaptive algorithm taken from [54], [55]. The next step in improving the behavior upon persistent losses is to estimate the average burst length and adapt the encoder's configuration. We argue that an appropriate configuration of the coding scheme (r and φ) would likely yield better performance when considering bursty links. Since our design is flexible enough to modify the coding strategy, we will broaden the analysis, to assess the potential impact of changing the coding configuration.

Section II-B introduces the interleaving of coded packets. However, interleaving blocks do not have to be a multiple of generation size or depend on coding. As shown in [66], interleaving can be applied in audio transmission over packets without any coding, distributing the impact of burst losses over non-consecutive audio samples. Depending on application data transmission rate, the latency introduced by building small interleaving blocks could be acceptable. Decoupling interleaving from FEC coding would allow using the basic XOR scheme with a nearly random distribution of coded packets in the transmission queue.

VIII. CONCLUSION

We presented the design and implementation of rQUIC, an integration of the QUIC protocol with a coding 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. After an extensive measurement campaign, the obtained results demonstrate that the proposed scheme not only yields a lower average delay, but it also brings a more predictable behavior, because the observed delays show less variability. 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.

In our future work, we will exploit the rQUIC implementation to evaluate the benefits of using more advanced coding strategies, which were already considered in our design. We will also focus on the relationship between the coding and congestion control mechanisms. In addition, we will evaluate the improvements that might be brought about by including additional functionalities. We will pay special attention to the use of multipath, and its integration with the coding module. In this sense, we argue that coded packets might be useful for various endpoints, which could recover different lost packets with the same coded packet.

REFERENCES

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, and J. Bailey, "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*. New York, NY, USA: ACM Press, 2017, pp. 183–196.

[2] *Usage Statistics of QUIC for Websites, August 2021*. Accessed: Aug. 2, 2021. [Online]. Available: https://w3techs.com/technologies/details/ce-quic

[3] J. Iyengar and M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, document RFC 9000, Internet Requests for Comments, RFC Editor, May 2021.

[4] R. Paulo. *Implementations Quicwg/Base-Drafts Wiki*. Accessed: Aug. 2, 2021. [Online]. Available: https://github.com/quicwg/base-drafts/wiki/Implementations
[5] F. Michel, Q. D. Coninck, and O. Bonaventure, "QUIC-FEC: Bringing the benefits of forward erasure correction to QUIC," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, May 2019, pp. 1–9.
[6] P. Garrido, I. Sanchez, S. Ferlin, R. Aguero, and O. Alay, "RQUIC: Integrating FEC with QUIC for robust wireless communications," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–7.
[7] Q. D. Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing QUIC," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 19–24.
[8] I. Swett, M.-J. Montpetit, V. Roca, and F. Michel, *Coding for QUIC*, document Internet-Draft Draft-Swett-Nwcrg-Coding-for-Quic-04, Internet Engineering Task Force, Mar. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-swett-nwcrg-coding-for-quic-04
[9] V. Roca, F. Michel, I. Swett, and M.-J. Montpetit, *Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for QUIC*, document Internet-Draft Draft-Roca-Nwcrg-Rlc-Fec-Scheme-for-Quic-03, Internet Engineering Task Force, Mar. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-roca-nwcrg-rlc-fec-scheme-for-quic-03
[10] J. Iyengar and I. Swett, *QUIC Loss Detection and Congestion Control*, document Internet-Draft Draft-Ietf-Quic-Recovery-29, Internet Engineering Task Force, Jun. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-recovery-29
[11] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2011, pp. 181–194.
[12] M. Thomson and S. Turner, *Using TLS to Secure QUIC*, document Internet-Draft Draft-Ietf-Quic-Tls-29, Internet Engineering Task Force, Jun. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-29
[13] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the QUIC protocol in Linux kernel," in *Proc. 21st ACM Int. Conf. Modeling, Anal. Simulation Wireless Mobile Syst.* New York, NY, USA: ACM, Oct. 2018, pp. 227–234.
[14] J. Iyengar and M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, document Internet-Draft Draft-Ietf-Quic-Transport-29, Internet Engineering Task Force, Jun. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-29
[15] J. Postel, *Transmission Control Protocol*, document STD 793, Internet Requests for Comments, RFC Editor, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc793.txt
[16] G. Albertazzi, M. Chiani, G. E. Corazza, A. Duverdier, H. Ernst, W. Gappmair, G. Liva, and S. Papaharalabos, *Forward Error Correction*. Boston, MA, USA: Springer, 2007, pp. 117–174. [Online]. Available: http://link.springer.com/10.1007/978-0-387-34649-6_4, doi: 10.1007/978-0-387-34649-6_4.
[17] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000. [Online]. Available: http://ieeexplore.ieee.org/document/850663/
[18] D. Katabi, S. Katti, and H. Rahul, "Harnessing network coding in wireless systems," in *Network Coding*. Boston, MA, USA: Academic, 2012, pp. 39–60. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780123809186000020
[19] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2003, p. 442.
[20] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.
[21] M. Wang and B. Li, "How practical is network coding?" in *Proc. 14th IEEE Int. Workshop Quality Service*, Jun. 2006, pp. 274–278. [Online]. Available: http://ieeexplore.ieee.org/document/4015763/
[22] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and T. Larsen, "Network coding for mobile devices–systematic binary random rateless codes," in *Proc. IEEE Int. Conf. Commun. Workshops*, no. 2, Jun. 2009, pp. 1–6.
[23] H. Liu, H. Ma, M. E. Zarki, and S. Gupta, "Error control schemes for networks: An overview," *Mob. Netw. Appl.*, vol. 2, no. 2, pp. 167–182, 1997, doi: 10.1023/A:1013676531988.
[24] S. Wunderlich, F. Gabriel, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar RLNC (CRLNC): A practical finite sliding window RLNC approach," *IEEE Access*, vol. 5, pp. 20183–20197, 2017.
[25] F. Gabriel, S. Wunderlich, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar RLNC with feedback (CRLNC-FB): Reducing delay in selective repeat ARQ through coding," *IEEE Access*, vol. 6, pp. 44787–44802, 2018.
[26] M. Zverev, P. Garrido, R. Agüero, and J. Bilbao, "Systematic network coding with overlap for IoT scenarios," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 1–6.
[27] Z. Huang, X. Wang, X. Chen, and H. Kan, "Network coding with interleaving," in *Proc. Int. Conf. Parallel Process. Workshops (ICPPW)*, Sep. 2007, pp. 1–6.
[28] D. Stolpmann, C. Petersen, V. Eichhorn, and A. Timm-Giel, "Extending On-the-fly network coding by interleaving for avionic satellite links," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–5.
[29] B. Adamson, C. Adjih, J. Bilbao, V. Firoiu, F. Fitzek, S. A. M. Ghanem, E. Lochin, A. Masucci, M.-J. Montpetit, M. V. Pedersen, G. Peralta, V. Roca, P. Saxena, and S. Sivakumar, *Taxonomy of Coding Techniques for Efficient Network Communications*, document RFC 8406, Jun. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8406.txt
[30] W. An, M. Médard, and K. R. Duffy, "Keep the bursts and ditch the interleavers," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6.
[31] J. Wu, B. Cheng, M. Wang, and J. Chen, "Priority-aware FEC coding for high-definition mobile video delivery using TCP," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 1090–1106, Apr. 2017.
[32] T. Tsugawa, N. Fujita, T. Hama, H. Shimonishi, and T. Murase, "TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee," in *Proc. Packet Video*, Nov. 2007, pp. 294–301.
[33] F. Teshima, H. Obata, R. Hamamoto, and K. Ishida, "TCP-TFEC: TCP congestion control based on redundancy setting method for FEC over wireless LAN," *IEICE Trans. Inf. Syst.*, vol. E100.D, no. 12, pp. 2818–2827, 2017.
[34] Y. Sato, H. Koga, and T. Ikenaga, "TCP using adaptive FEC to improve throughput performance in high-latency environments," *IEICE Trans. Commun.*, vol. E102.B, no. 3, pp. 537–544, 2019.
[35] N. Kuhn, E. Lochin, F. Michel, and M. Welzl, *Coding and Congestion Control in Transport*, document Internet-Draft Draft-Irtf-Nwcrg-Coding-and-Congestion-09, Internet Engineering Task Force, Jun. 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-nwcrg-coding-and-congestion-09
[36] P. Truchly, M. Sith, and R. Repka, "End-to-end packet loss differentiation algorithms and their performance in heterogeneous networks," in *Proc. 17th Int. Conf. Emerg. eLearning Technol. Appl. (ICETA)*, Nov. 2019, pp. 777–783. [Online]. Available: https://ieeexplore.ieee.org/document/9040082/
[37] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document RFC 3550, Jul. 2003. [Online]. Available: https://rfc-editor.org/rfc/rfc3550.txt
[38] A. H. Li, *RTP Payload Format for Generic Forward Error Correction*, document RFC 5109, Dec. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc5109.txt
[39] A. C. Begen, *RTP Payload Format for 1-D Interleaved Parity Forward Error Correction (FEC)*, document RFC 6015, Oct. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc6015.txt
[40] L. Liu and X. Dong, "Evaluating packet-level forward error correction: 1-D interleaved parity codes," in *Proc. 8th Int. Conf. Comput. Technol. Inf. Manage. (ICCM)*, vol. 1, Apr. 2012, pp. 370–375.
[41] Y. Go, H. Noh, G. Park, and H. Song, "Energy-efficient HTTP adaptive streaming with hybrid TCP/UDP over heterogeneous wireless networks," in *Proc. IEEE 20th Int. Symp. World Wireless, Mobile Multimedia Networks (WoWMoM)*, Jun. 2019, pp. 1–10.
[42] I. Swett. (2016). *QUIC FEC V1*. Accessed: Aug. 6, 2021. [Online]. Available: https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit
[43] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Medard, "Network coded TCP (CTCP)," 2013, *arXiv:1212.2291*. [Online]. Available: https://arxiv.org/abs/1212.2291
[44] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proc. IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5688180/
[45] J. Ridgewell and H. Elaarag, "NCTCP: A network coded TCP protocol," *Simul. Ser.*, vol. 48, no. 3, pp. 39–46, 2016.

[46] K. Alferaidi, R. Piechocki, and F. Alfordy, "Improving TCP performance in multi-hop coded wireless networks," in *Proc. 2nd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS)*, May 2019, pp. 1–6.

[47] B. Y. L. Kimura, D. C. S. F. Lima, and A. A. F. Loureiro, "Packet scheduling in multipath TCP: Fundamentals, lessons, and opportunities," *IEEE Syst. J.*, vol. 15, no. 1, pp. 1445–1457, Mar. 2021.

[48] V. Sharma, S. Kalyanaraman, K. Kar, K. K. Ramakrishnan, and V. Subramanian, "MPLOT: A transport protocol exploiting multipath diversity using erasure codes," in *Proc. 27th Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2008, pp. 121–125.

[49] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 465–478, Apr. 2015.

[50] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, 2017, pp. 160–166.

[51] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: A deployable multipath transport protocol," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[52] Q. D. Coninck and O. Bonaventure, *Multipath Extensions for QUIC (MP-QUIC)*, document Internet-Draft Draft-Deconinck-Quic-Multipath-07, Internet Engineering Task Force, May 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-07

[53] Y. Liu, Y. Ma, C. Huitema, Q. An, and Z. Li, *Multipath Extension for QUIC*, document Internet-Draft Draft-Liu-Multipath-Quic-03, Internet Engineering Task Force, Mar. 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-liu-multipath-quic-03

[54] S. Ferlin and O. Alay, "TCP with dynamic FEC for high delay and lossy networks," in *Proc. Int. Conf. Emerg. Netw. EXperiments Technol. (CoNEXT) Student Workshop*, 2016, pp. 1–3.

[55] S. Ferlin, S. Kucera, H. Claussen, and O. Alay, "MPTCP meets FEC: Supporting latency-sensitive applications over heterogeneous networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2005–2018, Oct. 2018.

[56] M. Thomson, *Version-Independent Properties of QUIC*, document Internet-Draft Draft-Ietf-Quic-Invariants-09, Internet Engineering Task Force, Jun. 2020. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-invariants-09

[57] V. Adat Vasudevan, C. Tselios, and I. Politis, "On security against pollution attacks in network coding enabled 5G networks," *IEEE Access*, vol. 8, pp. 38416–38437, 2020.

[58] Z. Cataltepe and P. Moghe, "Characterizing nature and location of congestion on the public Internet," in *Proc. 8th IEEE Symp. Comput. Commun. (ISCC)*, vol. 2, Jul. 2003, pp. 741–746.

[59] *Information Technology—Dynamic Adaptive Streaming Over HTTP (DASH)—Part 1: Media Presentation Description and Segment Formats*, Int. Org. Standardization, Geneva, Switzerland, Dec. 2019.

[60] D. Raca, M. Manifacier, and J. J. Quinlan, "GoDASH—GO accelerated HAS framework for rapid prototyping," in *Proc. 12th Int. Conf. Qual. Multimedia Exper. (QoMEX)*, May 2020, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9123103/

[61] A. Raake, M.-N. Garcia, W. Robitza, P. List, S. Goring, and B. Feiten, "A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1," in *Proc. 9th Int. Conf. Qual. Multimedia Exper. (QoMEX)*, May 2017, p. 1203.

[62] S. Yang and R. W. Yeung, "Batched sparse codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5322–5346, Sep. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6847232/

[63] G. Fairhurst, A. Custura, and T. Jones, *Changing the Default QUIC ACK Policy*, document Internet-Draft Draft-Fairhurst-Quic-Ack-Scaling-04, Internet Engineering Task Force, Mar. 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-fairhurst-quic-ack-scaling-04

[64] J. Iyengar and I. Swett, *QUIC Acknowledgement Frequency*, document Internet-Draft Draft-Ietf-Quic-Ack-Frequency-00, Internet Engineering Task Force, Jul. 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-ack-frequency-00

[65] K. V. Cardoso and J. F. D. Rezende, "Accurate hidden Markov modeling of packet losses in indoor 802.11 networks," *IEEE Commun. Lett.*, vol. 13, no. 6, pp. 417–419, Jun. 2009.

[66] C. Perkins, O. Hodson, and V. Hardman, "A survey of packet loss recovery techniques for streaming audio," *IEEE Netw.*, vol. 12, no. 5, pp. 40–48, Sep. 1998.

MIHAIL ZVEREV received the degree in telecommunications engineering from the University of Cantabria, Santander, Spain, in 2014, and the M.Sc. degree in project management from La Salle, University of Ramón Llull, Barcelona, in 2017. He is currently pursuing the joint Ph.D. degree with Ikerlan Technology Research Centre in collaboration with the University of Cantabria. His area of work is mainly centered around reducing latency in transport layer protocols.

PABLO GARRIDO received the degree in telecommunications engineering, the M.Sc. degree in mathematics and computing, and the Ph.D. degree in information technologies from the University of Cantabria, in 2013, 2014, and 2018, respectively. From 2018 to 2021, he was a Research and Development Engineer at the IoT Cybersecurity Department, Ikerlan Technology Research Center and currently the Innovative Solutions Architect at Nemergent Solutions. He has published more than 20 papers in peer-review conferences and journals in his research fields. His area of work is mainly centered on network coding, transport protocols, the IoT, and mission critical solutions.

FÁTIMA FERNÁNDEZ received the degree in telecommunication technologies engineering and the M.Sc. degree in telecommunication engineering from the University of Cantabria, Santander, Spain, in 2017 and 2019, respectively. She is currently pursuing the joint Ph.D. degree with Ikerlan Technology Research Centre in collaboration with the University of Cantabria. In 2019, she was a Fellow Researcher with the Communications Engineering Department, University of Cantabria. Her area of work is mainly centered around congestion control algorithms for transport layer protocols.

JOSU BILBAO (Senior Member, IEEE) received the degree in telecommunication engineering from the Faculty of Engineering of Bilbao (UPV/EHU), Spain, the M.Sc. degree in communications and control from the University of the Basque Country, Spain, and the Ph.D. degree in computer science from the University of Navarra, Spain. He has been with Ikerlan Technology Research Centre, Arrasate/Mondragón, Spain, since 2003, where he is the Head of the Information and Communication Technologies Area. His current research interests include reliable communications, real-time cyber-physical systems integration in the Internet of Things (IoT), fog-based the IoT architectures, 5G, artificial intelligence, and edge computing. He has taken part in different standardization bodies and platforms, such as IETF, IEEE, HANA, and ETG.

ÖZGÜ ALAY (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical and electronic engineering from Middle East Technical University, Turkey, and the Ph.D. degree in electrical and computer engineering from Tandon School of Engineering, New York University, USA. She is currently an Associate Professor with the University of Oslo, Norway, and also the Head of Mobile Systems and Analytics (MOSAIC) Department, Simula Metropolitan, Norway. She is the author of more than 70 peer-reviewed publications. Her research interests include 5G networks, multi-connectivity and multipath protocols, the IoT, drone communications, and multimedia systems for future mobile networks.

SIMONE FERLIN received the Dipl.-Ing. degree in information technology with major in telecommunications from Friedrich-Alexander Erlangen Nuernberg University, Germany, in 2010, and the Ph.D. degree in computer science from the University of Oslo, Norway, in 2017. Her research interests include intersection of cellular networks and the Internet, with her research focusing on QoS and cross-layer design, transport protocols, congestion control, network performance, security, and measurements. Her dissertation focused on improving robustness in multipath transport for heterogeneous networks with MPTCP. She is currently a Software Architect at Ericsson AB. She also actively serves on technical boards of major conferences and journals in these areas.

ANNA BRUNSTROM (Member, IEEE) received the B.Sc. degree in computer science and mathematics from Pepperdine University, CA, USA, in 1991, and the M.Sc. and Ph.D. degrees in computer science from the College of William and Mary, VA, USA, in 1993 and 1996, respectively. She joined the Department of Computer Science at Karlstad University (KAU), Sweden, in 1996, where she is currently a Full Professor and a Research Manager of the Distributed Systems and Communications Research Group. Her research interests include internet architectures and protocols, techniques for low latency internet communication, multipath communication, performance evaluation, and optimization of mobile broadband systems, including 5G. She is currently the KAU Principal Investigator within the EU H2020 Project 5GENESIS. She is the Co-Chair of the RTP Media Congestion Avoidance Techniques (rmcat) working group within the IETF. She has authored/coauthored over 200 international journals and conference papers.

RAMÓN AGÜERO (Senior Member, IEEE) received the M.Sc. degree (Hons.) in telecommunications engineering from the University of Cantabria, Santander, Spain, in 2001, and the Ph.D. degree (Hons.), in 2008. Since 2016, he has been the Head of the IT area (deputy CIO) at the University of Cantabria. He is currently an Associate Professor with the Communications Engineering Department, University of Cantabria. His research interests include future network architectures, especially regarding the (wireless) access part of the network and its management. He is interested on multihop (mesh) networks and network coding. He has published more than 200 scientific articles in such areas. He has supervised five Ph.D. students and more than 70 B.Sc. and M.Sc. theses. He is the main instructor in courses dealing with networks and traffic modeling at B.Sc. and M.Sc. levels. He is a regular TPC member and a reviewer on various related conferences and journals. He has been serving in the Editorial Board of IEEE Communication Letters as a Senior Editor, since 2019. He serves in the Editorial Board of IEEE Open Access Journal of the Communications Society, *Wireless Networks* (Springer), and *Mobile Information Systems* (Hindawi).