QUIC-FEC: Adding Forward Erasure Correction to QUIC

Design, implementation and experiments

François Michel, Quentin De Coninck, Olivier Bonaventure

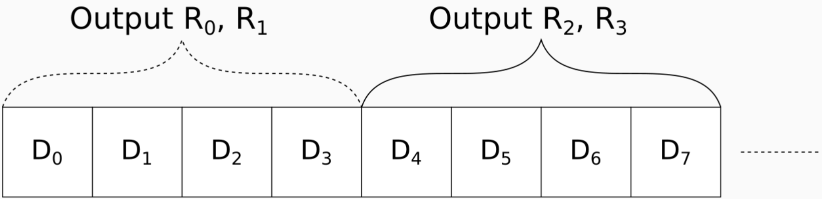
May 7, 2019

UCLouvain, Louvain-la-Neuve, Belgium

Forward Erasure Correction (FEC)

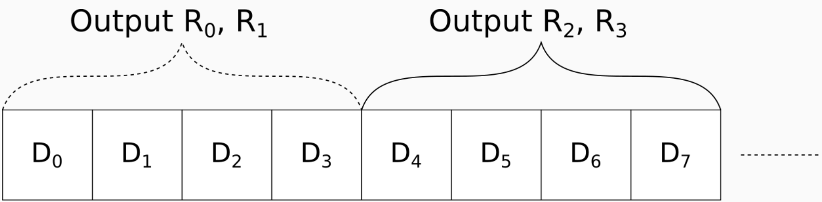
- Sending redundant data (repair symbols) along with the important data (source symbols), before the source symbols are marked as lost
- Allows to recover lost data without waiting for retransmissions
- Useful for short transfers or real-time communications in lossy/high delay scenarios
- Often requires additional bandwidth compared to retransmissions, because we don't know which packet will be lost.

A small example



Sent sequence :

A small example



Sent sequence :

Adding Forward Erasure Correction to QUIC: two approaches

Application-based FEC

- Using one or more streams for the application payload
- Using dedicated stream(s) (or datagrams ?) to send the redundancy
- Only protects the application traffic

Pushes the complexity to the application, but small control on how the redundancy is packetised

Providing FEC at the transport level

- Using separate frames to send the redundancy and user data
- Better control on how the data are packetised
- Can be used by any application
- Can protect more than the application traffic: (flow control frames, ...)

Our work with QUIC and FEC

We worked on a design and implementation of QUIC+FEC to study the benefits of FEC with QUIC.

- First implementation using `quic-go` (Google-QUIC)
- Second implementation using `picoquic` (IETF-QUIC, draft 14)

Our implementations seamlessly allow the use of three different coding algorithm: XOR, Reed-Solomon and (convolutional) Random Linear Codes.

Defining Source Symbols

FEC protects source symbols by sending redundancy (repair symbols). We first have to define what are the source symbols.

- It could be stream chunks of equal size (stream-based protection)
  - No overhead but only protects application stream data
- It could be QUIC packets (packet-based protection)
  - Additional overhead (ex: stream frames headers), but allows to protect more than stream data (DATAGRAM frames ?)

We chose the packet-based approach.

Sending the repair symbols

We define a new QUIC frame to transport repair symbols, the FEC frame.

<table>
<tr><td><code>type byte</code></td><td colspan="2">Data Length (15)</td><td>F</td><td>Offset (8)</td></tr>
<tr><td colspan="5">Repair FEC Payload ID (64)</td></tr>
<tr><td>N. S. S. (8)</td><td>N. R. S. (8)</td><td colspan="3"></td></tr>
<tr><td colspan="5">Repair Symbol Payload
...</td></tr>
</table>

Figure 1: Wire format of a FEC frame. The Repair FEC Payload ID field is opaque to the protocol and is populated by the underlying FEC Scheme.

What to do when recovering a packet ?

- ACKing a recovered packet could send a confusing signal to the sender: if the loss is due to congestion, the congestion window won't be adapted
- Not ACKing a recovered packet would lead to a retransmission of its content
- We propose to explicitly signal that a packet has been recovered through a dedicated frame (the RECOVERED frame)
  - Currently, similar format to an ACK frame but announces which are the recovered packets

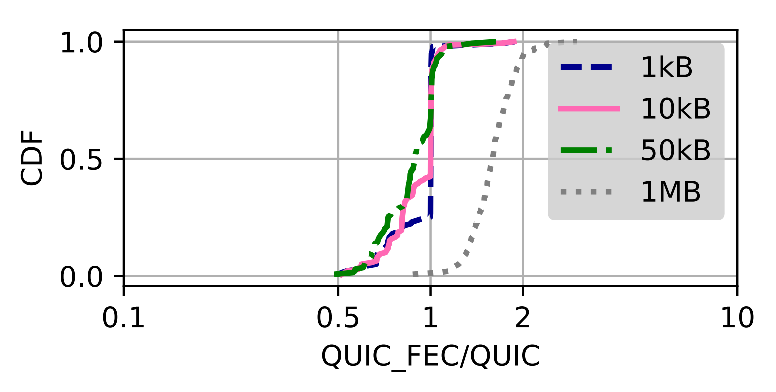
Some results of our implementations

- Simple request-response use-case with different file sizes, using Mininet
- We use a seeded loss generator
- Experiment parameters based on in-flight communications[1] (high delays, high loss rate)
- Still some non-determinism in the experiments (`quic-go` uses several threads)

[1] Results based on a study of Rula *et al.*
`http://www.cs.northwestern.edu/~jpr123/papers/www-flight.pdf`

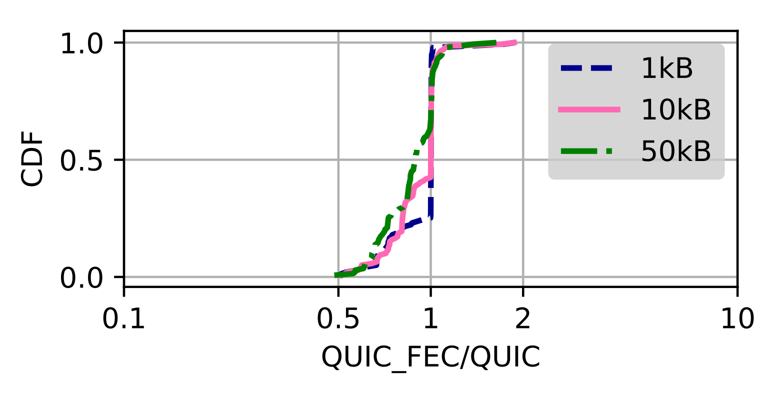
Some results of our implementations

(30, 20) Reed-Solomon code (10 repair symbols for 20 source symbols)



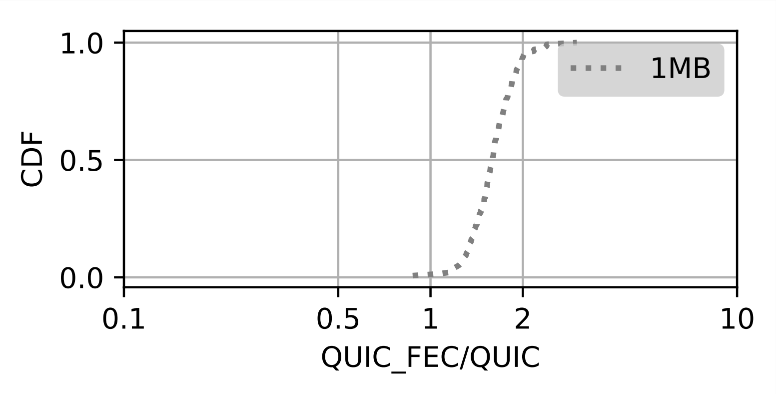
Some results of our implementations

Small HTTP responses are highly impacted by tail losses. FEC can help for that kind of request-response use-cases



Some results of our implementations

The impact of a tail loss on a larger transfer is small compared to the total time needed to transfer the additional redundancy

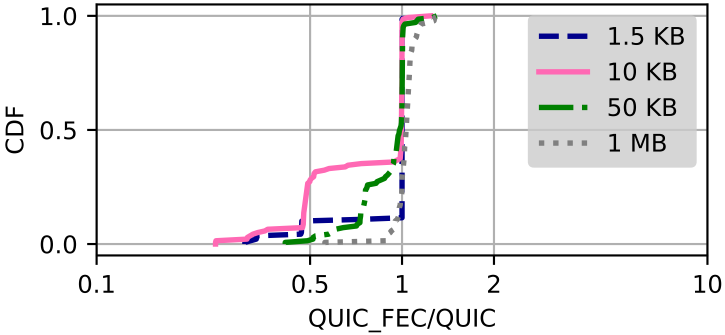


Only protecting the end of the download

- Some early results using `picoquic`, only protecting the end of the download, to reduce the negative impact of redundancy
- The `quic-go` and `picoquic` results are not directly comparable in details: the DCT has been computed slightly differently, the designs are slightly different, ...

Only protecting the end of the download

Only protecting the end of the download reduces the negative impact of FEC. There is still a small control overhead.



Some results of our implementations

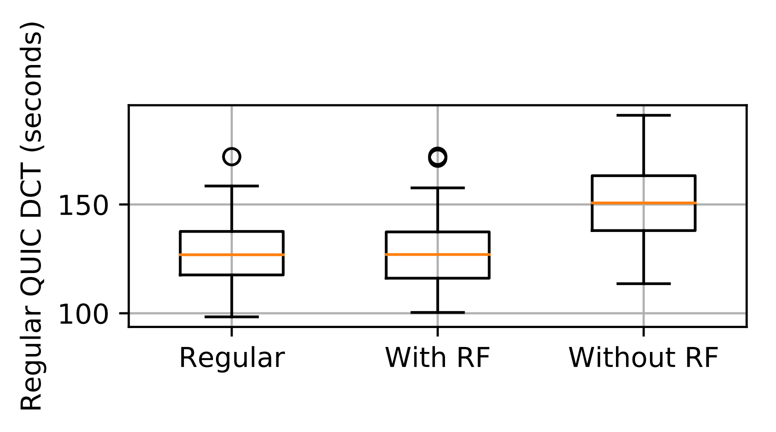
We studied how QUIC+FEC behaves when competing with QUIC.

Scenario:

- long bulk download background traffic
- 3 candidates for the background traffic
  - Regular QUIC
  - QUIC+FEC sending RECOVERED frames when packets are recovered
  - QUIC+FEC simply acknowledging the recovered packets
- We study the Download Completion Time for a regular QUIC foreground traffic when competing with each of these background traffics
- No medium losses applied to the communication (the detected losses are all due to congestion)

Some results of our implementations

The RECOVERED frames ensure to avoid being unfair when competing with regular QUIC flows



But we may need more experimental results to confirm this

Conclusion

- Using packets as source symbols enables the protection of other frames than STREAM frames (DATAGRAM, ...)
- FEC with QUIC also has an interest for short request-response scenario
- Recovering packets should be done carefully w.r.t. congestion control
- We would like to experiment in the wild (also with real-time use-cases)