QUIC-FEC: Adding Forward Erasure Correction to QUIC

Impementation and experiments

François Michel, Quentin De Coninck, Olivier Bonaventure

March 28, 2019

UCLouvain, Louvain-la-Neuve, Belgium

QUIC: Reliable, stream based transport protocol

- Provides reliable, encrypted and authenticated transport
- Provides stream multiplexing to avoid head-of-line blocking
- The reliable transport is provided using retransmissions
- FEC was part of the early versions of QUIC. It has now been postponed.

QUIC packets

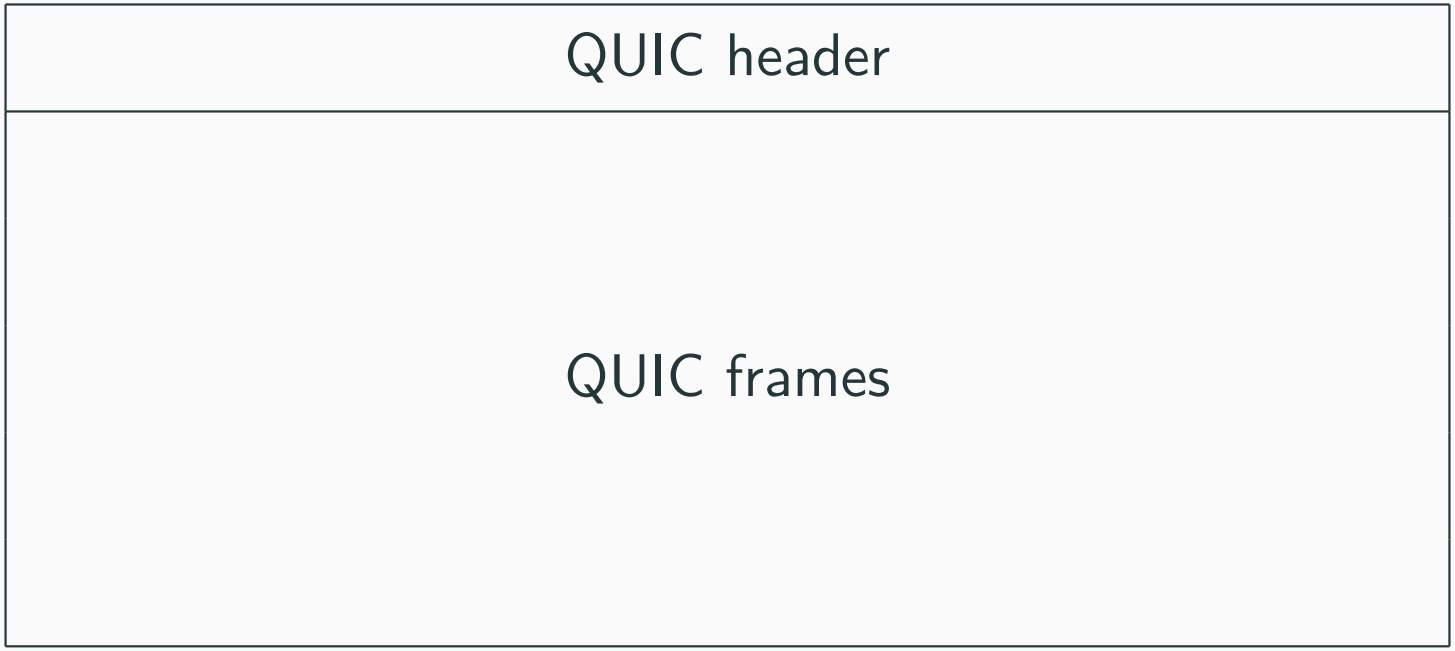


Figure 1: A QUIC packet is a container for frames.

QUIC packets

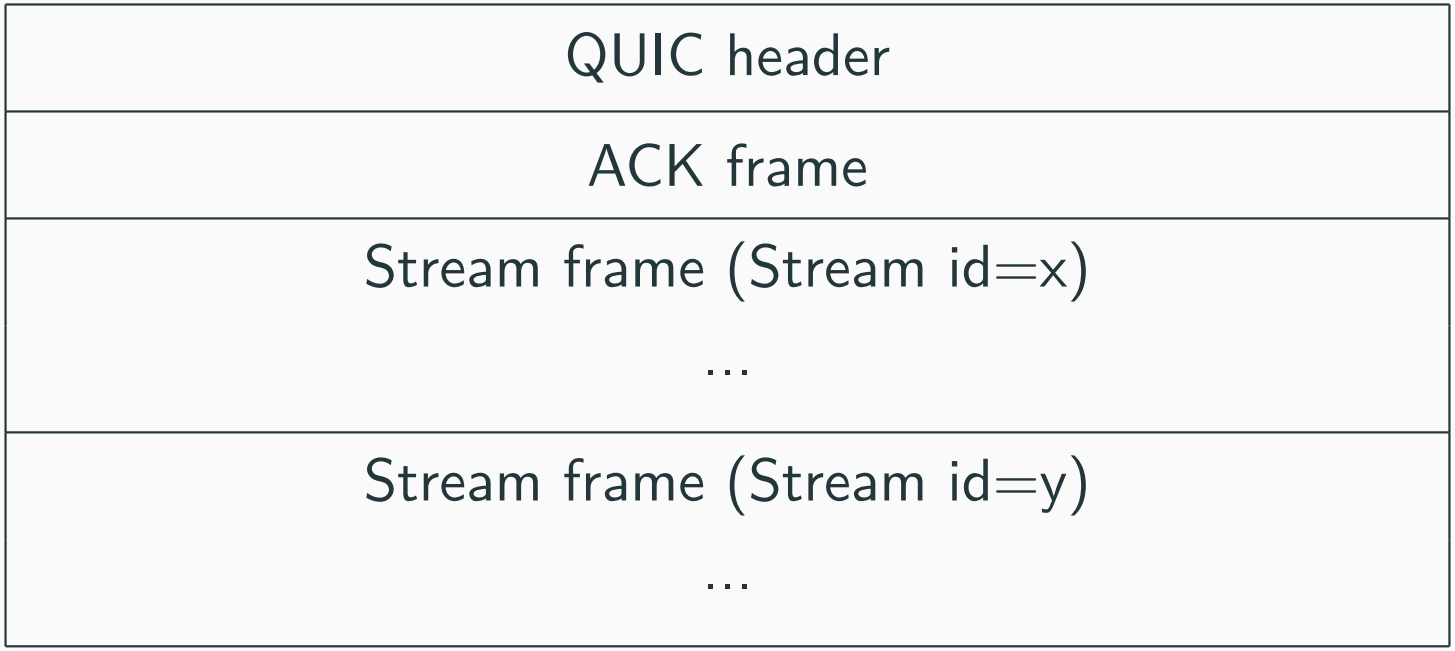


Figure 2: Stream frames contain user data, ACK frames contain SACK

Our work with QUIC and FEC

draft-swett-nwcrg-coding-for-quic already proposes a design for QUIC+FEC<. We worked on a design and implementation of QUIC+FEC concurrently to the draft elaboration.

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

Some of our choices differ from the current NWCRG draft. Let's discuss these points.

Disclaimer: some frame formats will be shown. We are only presenting our implementations, not proposing them as a draft.

What to protect - draft-swett-nwcrg-coding-for-quic

Protect stream chunks of fixed size E.

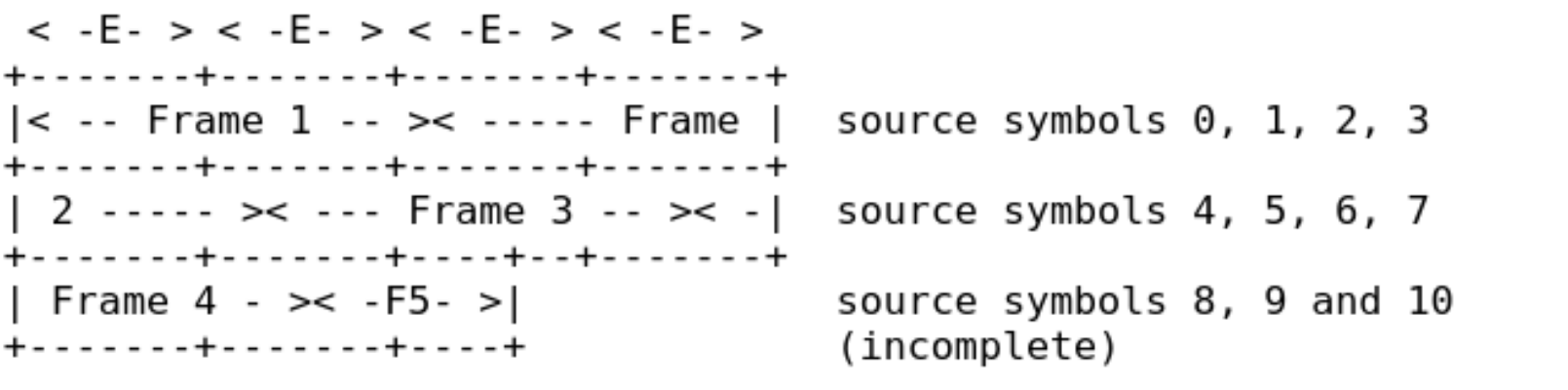


Figure 2: Example of source symbol mapping, when the E value is relatively small.

draft-swett-nwcrg-coding-for-quic - Pros

Protect stream chunks of fixed size E.

- No signaling needed to announce the source symbols
- No control overhead. The source symbols only contain stream data (no stream frame header, ...)

Protecting a packet - Our approach

Our implementation(s) have been developed in parallel to the draft. Our work is highly inspired by FECFRAME (RFC 6363)

We currently protect the frames transported in packets.

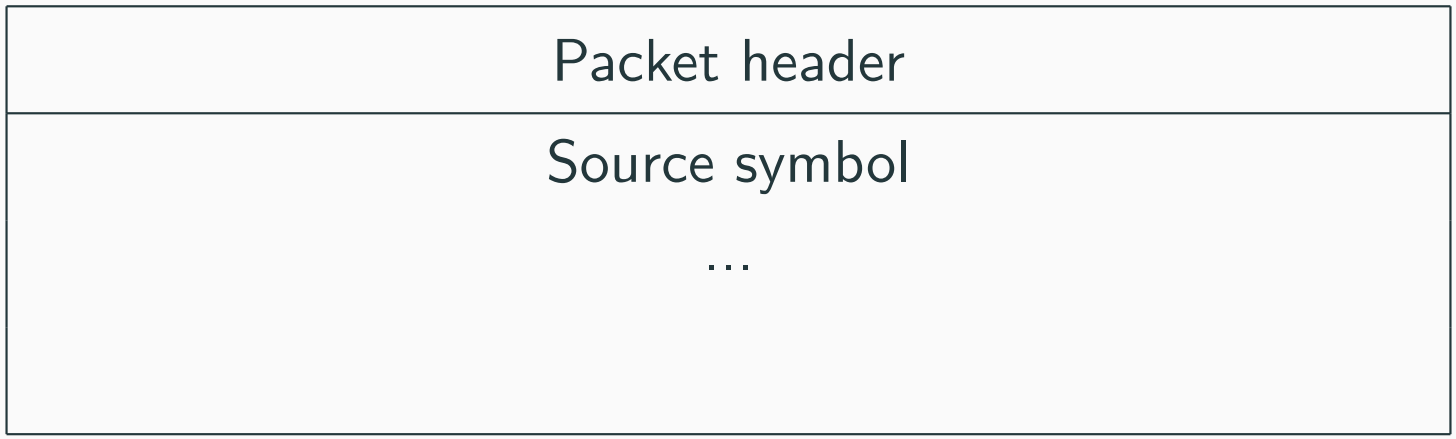


Figure 3: A source symbol in a QUIC packet.

Our approach - Pros

- No fixed source symbols size is required
  - QUIC naturally handles padding: adding zero-padding to a source symbol is identical to adding PADDING frames.
    - *! The padding does not need to be effectively sent !*
  - Solves the silent-period problem: we don't need to fill source symbols with user data to protect them
- Solves the problem of source symbols spanning two different packets

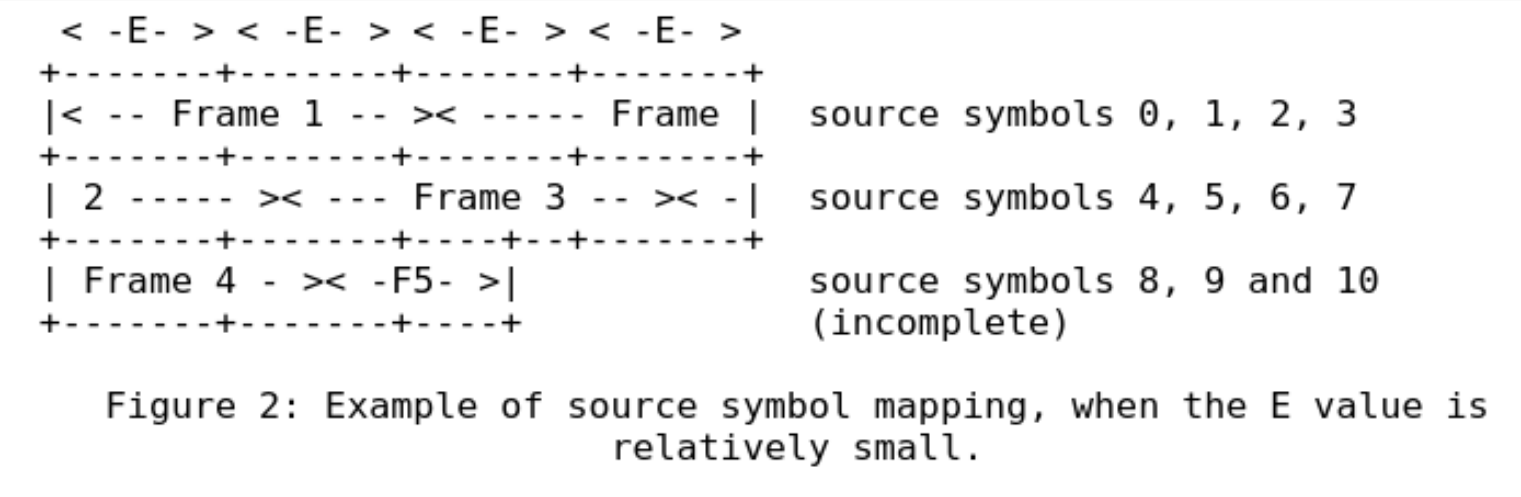


Figure 2: Example of source symbol mapping, when the E value is relatively small.

Our approach - Pros

- We can protect more than just STREAM data.
  - Flow control frames ?
  - Future DATAGRAM/MP/UNRELIABLE STREAM frames ?
- (The server and client can agree on which frames to protect inside a packet and filter-out non-critical frames from the source symbols)

Our approach - Cons

- Need explicit signalling to identify the source symbols
  - Add a new frame in the packet to protect ? (our approach in IETF-QUIC)

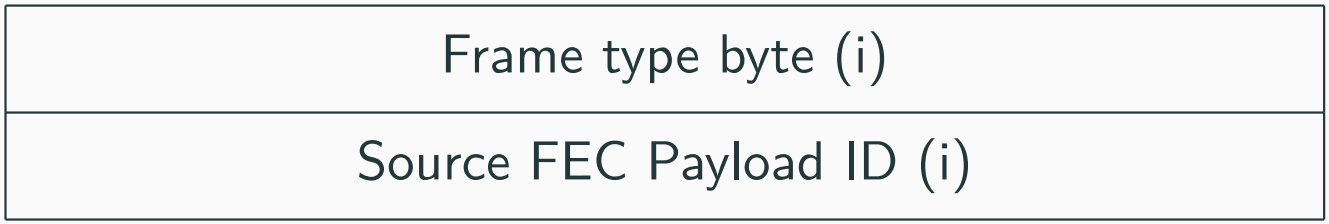


Figure 4: The `SFPID` (Source FEC Payload ID) frame format. The Source FEC Payload ID field is opaque (chosen by the FEC Scheme).

Our approach - Cons

- Increased overhead :'(
  - We protect the frames and their header too
  - **If a protected packet has the MTU size, the repair symbols won't likely fit into one packet !**
  - In our implementation, we slightly restrict the size of a packet payload to ensure the repair symbols to fit into a single packet.

Sending the repair symbols

Similarly to draft-swett-nwcrg-coding-for-quic, we add a frame to transport repair symbols, the FEC frame.

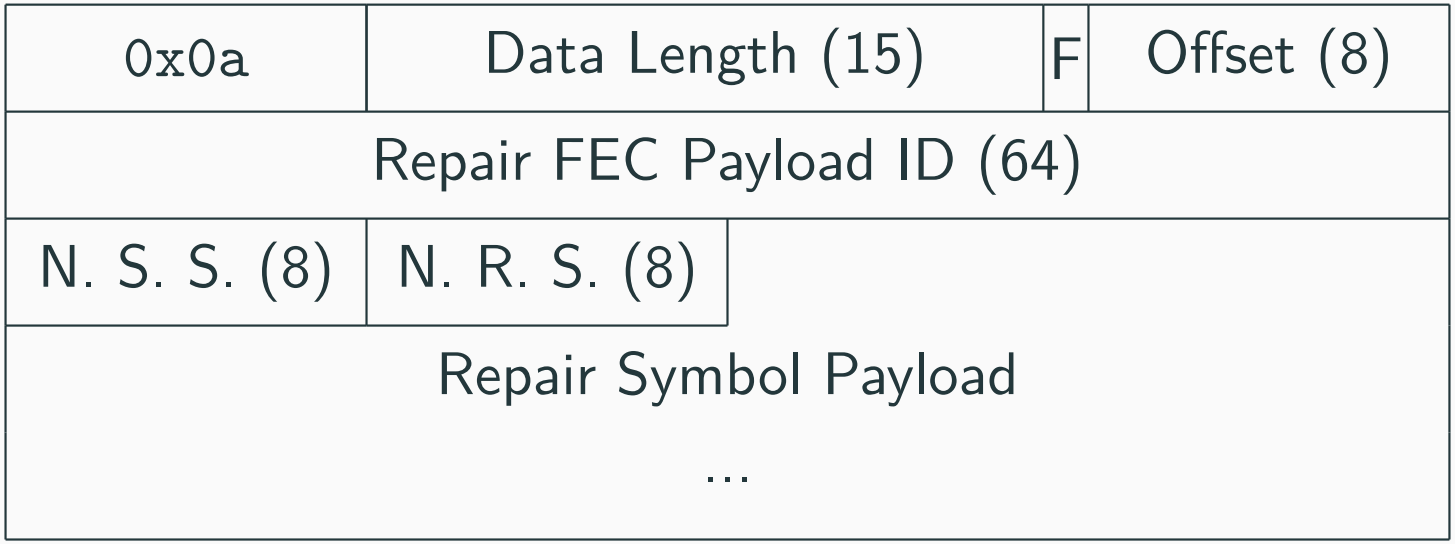


Figure 5: Wire format of a FEC frame. The Repair FEC Payload ID field is opaque to the protocol and is populated by the underlying FEC Scheme.

Explicitly signalling a packet recovery

- 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, identical format to an ACK frame but announces which are the recovered packets

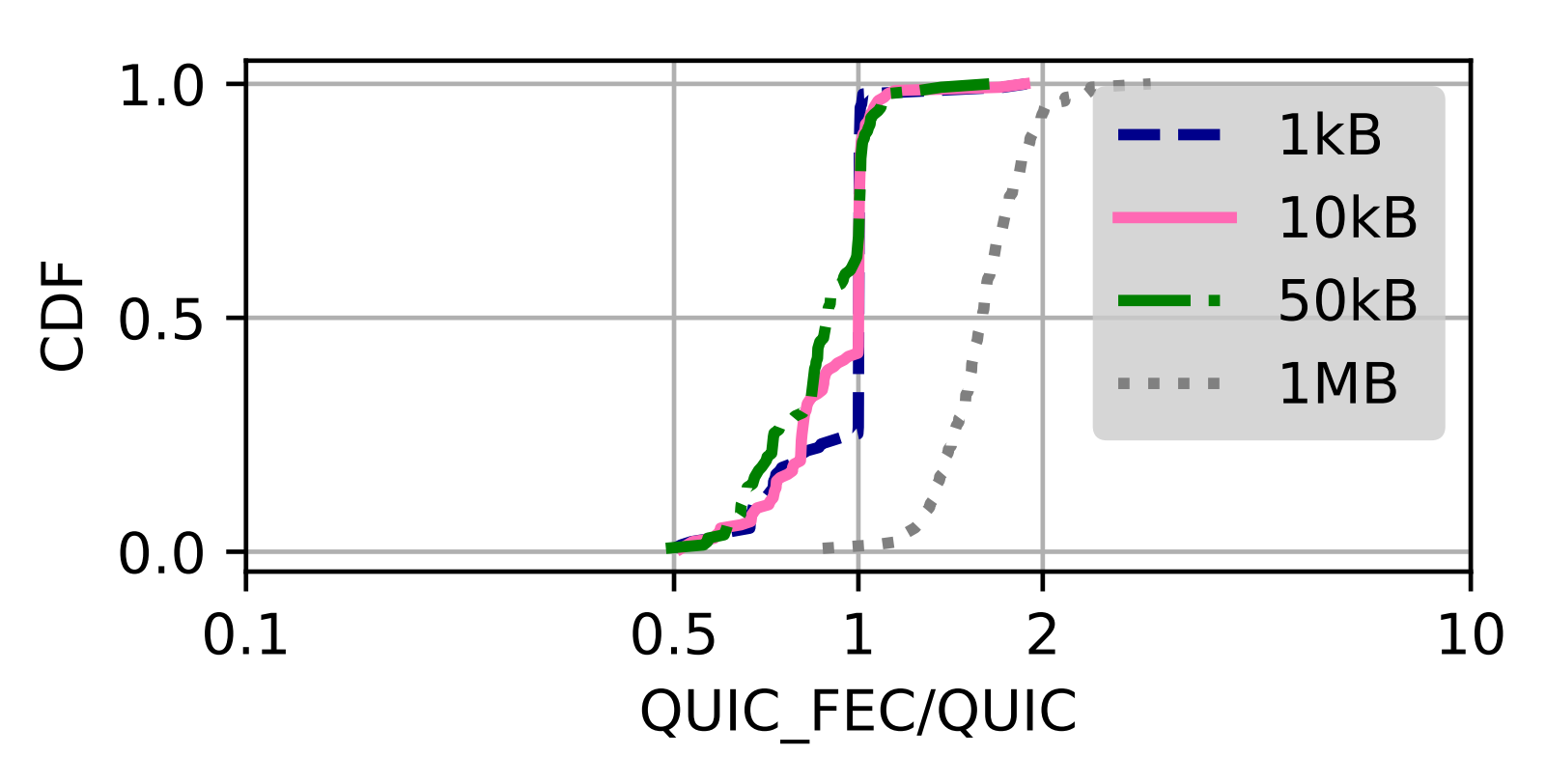
Some results of our implementations

- 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`

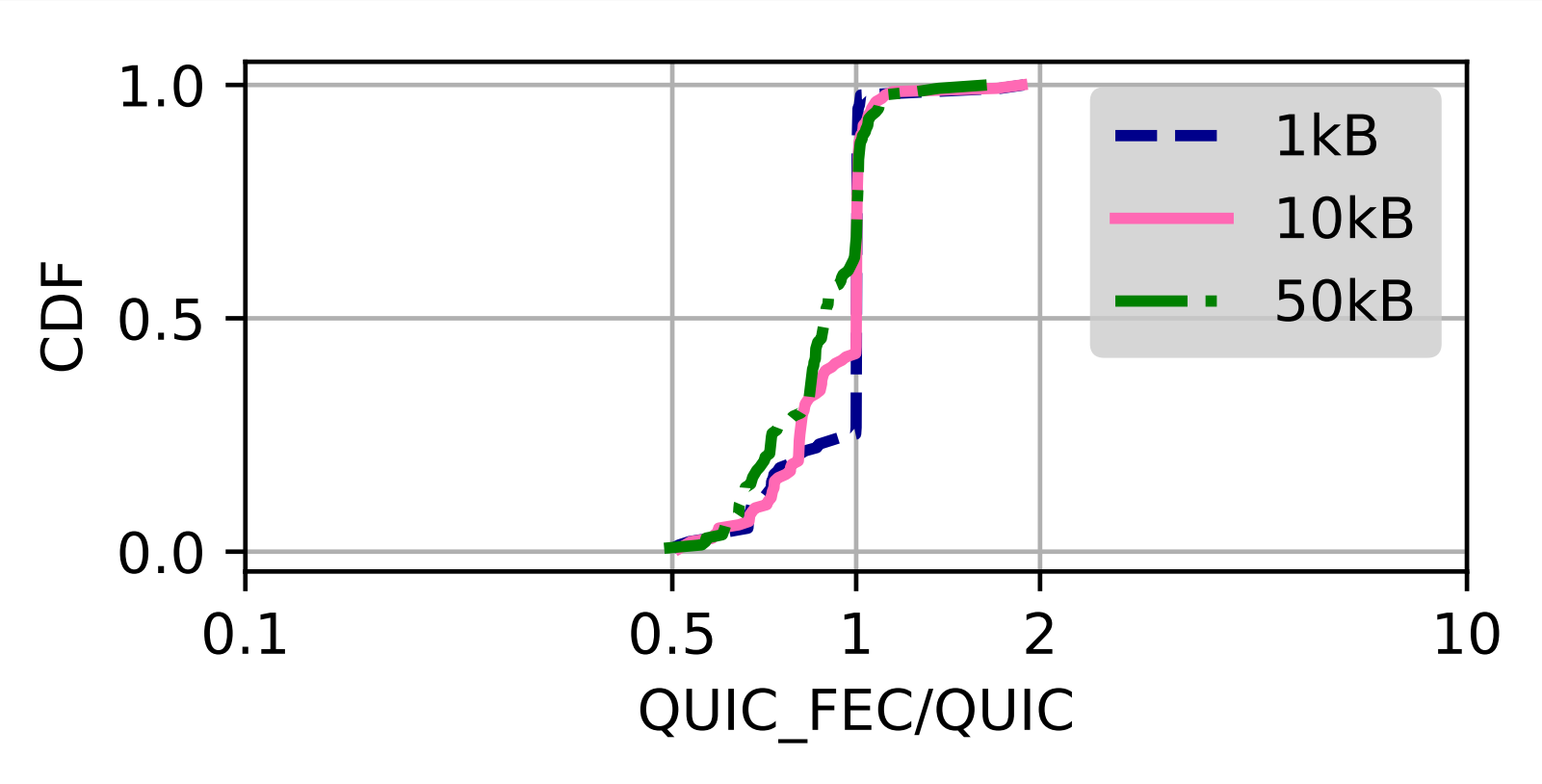
Some results of our implementations

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



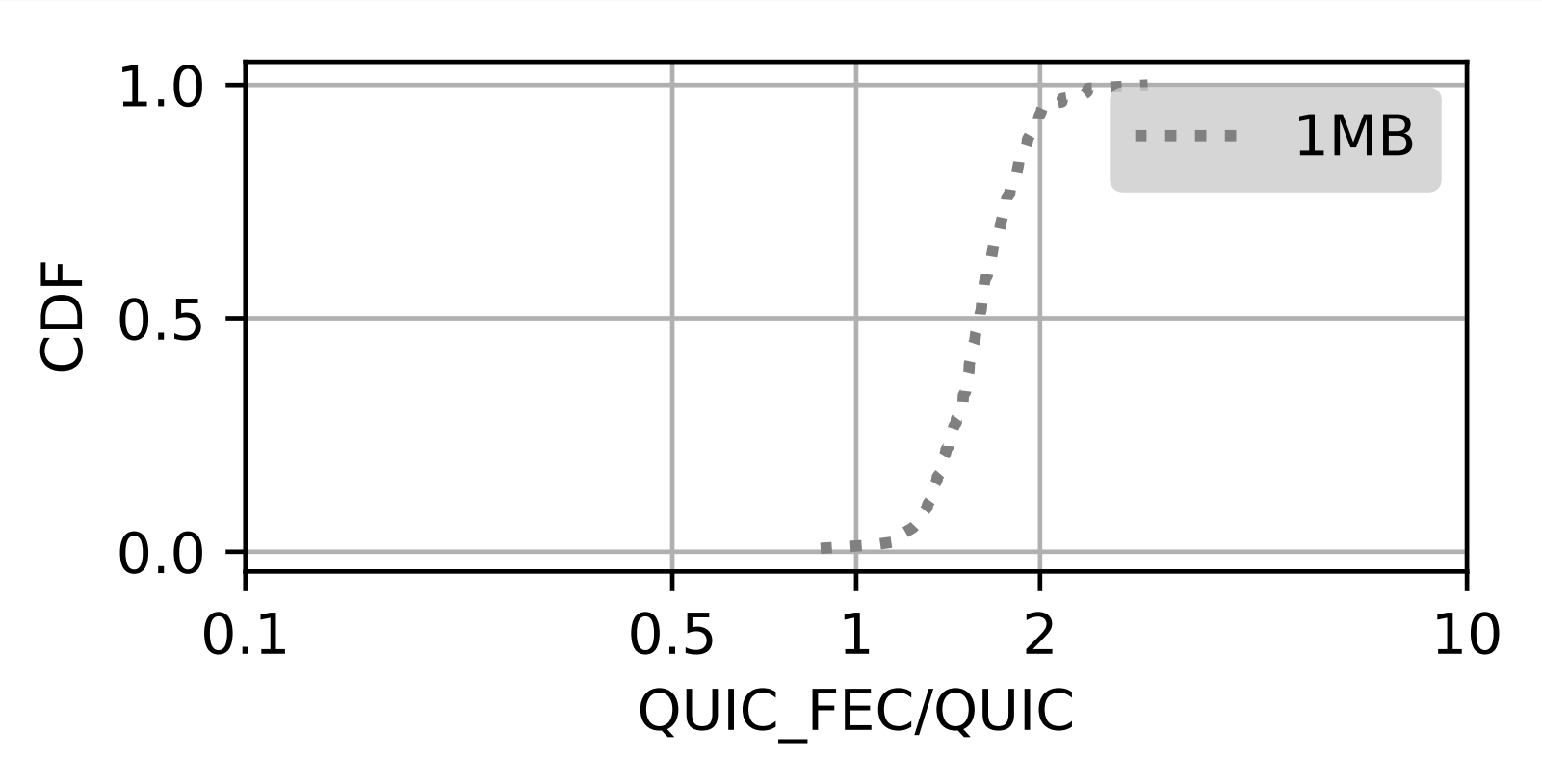
Some results of our implementations

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

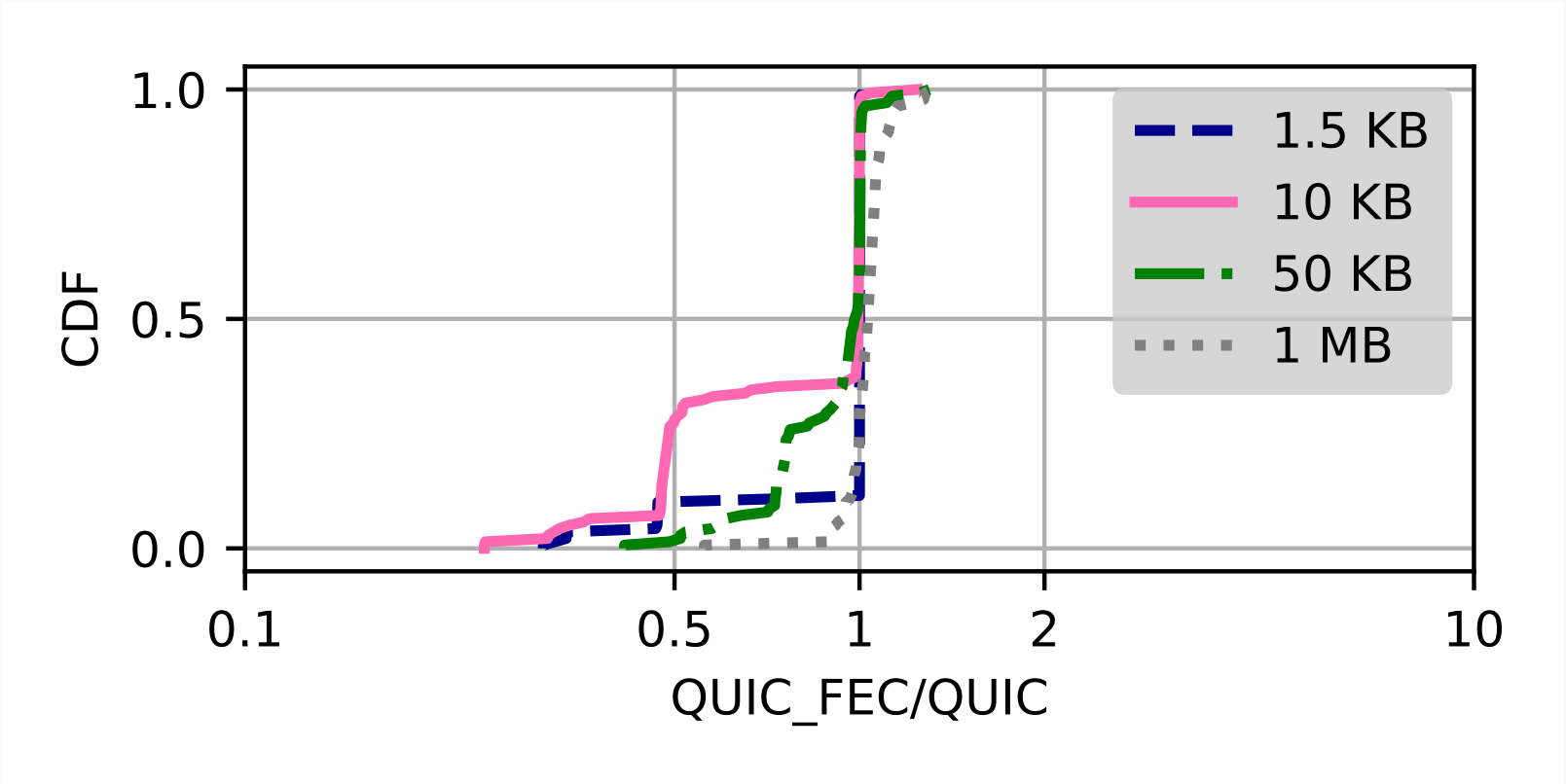


Only protecting the end of the download

- 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

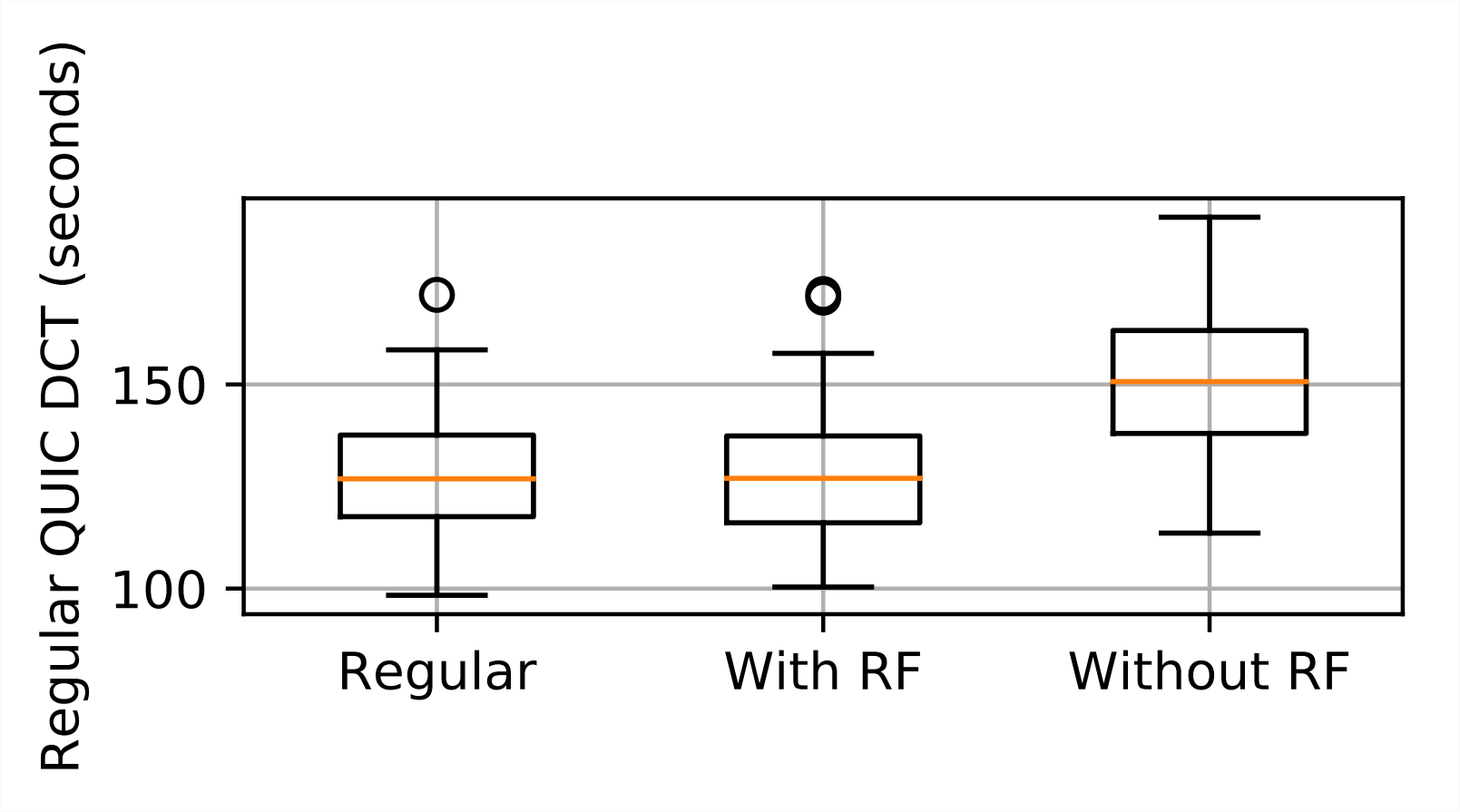
We studied how QUIC+FEC behaves when competing with QUIC.

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

- FEC with QUIC also has an interest for short request-response scenario
- Both stream-chunk-based and packet-based have their advantages
- The packet-based approach enables the protection of other frames than STREAM frames (DATAGRAM, ...)
- 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)