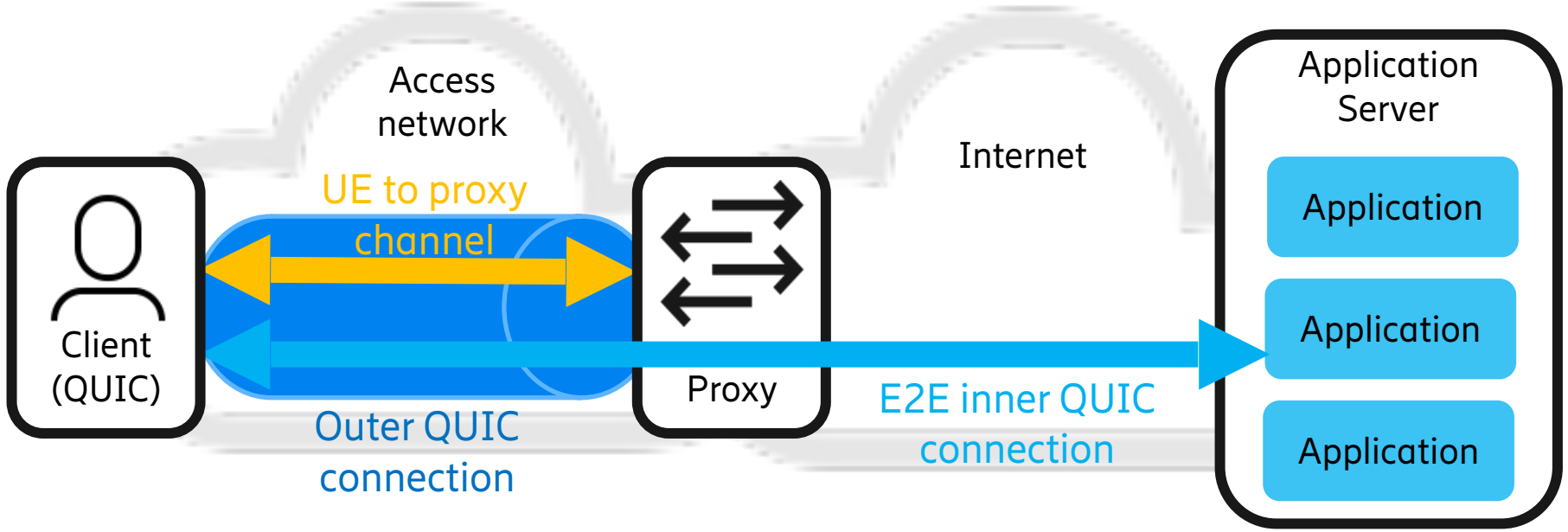
Evaluation of QUIC-based MASQUE Proxying

Mirja Kühlewind, Matias Carlander-Reuterfelt, Marcus Ihlar, Magnus Westerlund

CoNEXT EPIQ Workshop 2021

Proxy setup using MASQUE for QUIC tunneling

- The IETF MASQUE working group develops extensions to the `HTTP CONNECT` methods
- HTTP/3 as a tunneling protocol to forward UDP or IP packets between a client and a proxy
- Two tunnel modes: unreliable **datagram mode** or reliable **stream mode**



Our study

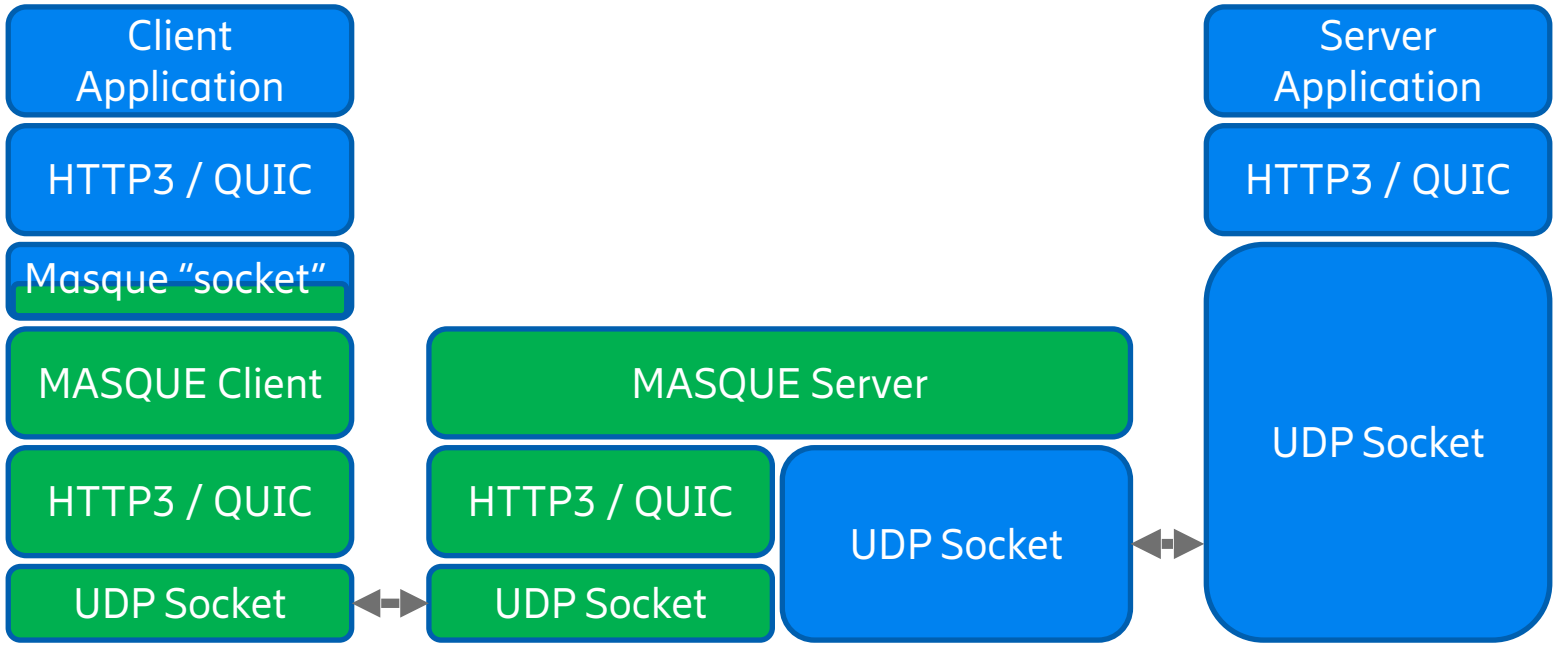
1. Implementation of MASQUE in `aioquic` and integration into a docker based emulation setup
   - Evaluation is restricted to a single implementation since there are few stacks that currently support MASQUE

2. Evaluation of MASQUE under basic network condition
   - Investigate different delay and loss scenarios as well as use of different QUIC packet sizes
   - Quantify overhead of adding an additional encapsulation and impact of nested congestion control

→ This study provides general findings about implications of the use of QUIC as a tunnel encapsulation
   - It do not provide a full performance assessment
   - But can provide valuable input to the on-going MASQUE standardization process

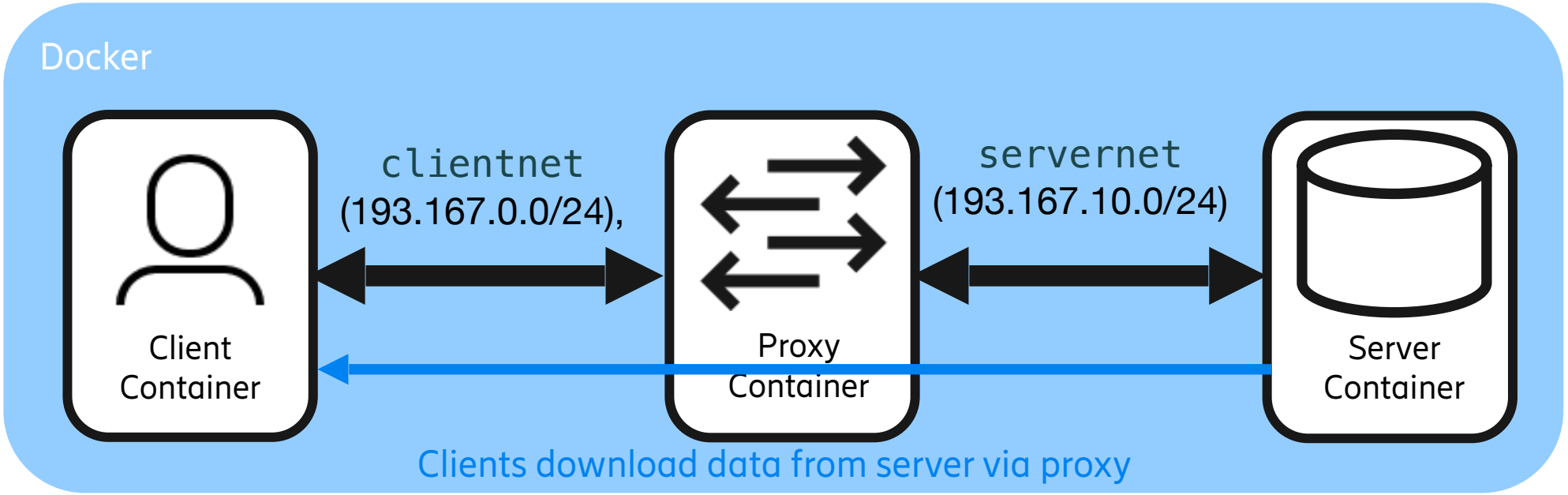
MASQUE Application structure

- **CONNECT-UDP method** to transform an HTTP request stream into a tunnel for UDP payload
- **HTTP datagrams** to multiplex unreliable delivery of payload data using QUIC datagrams
- **CAPSULE frames** to e.g. register the use of datagrams will be transmitted reliably over a QUIC stream



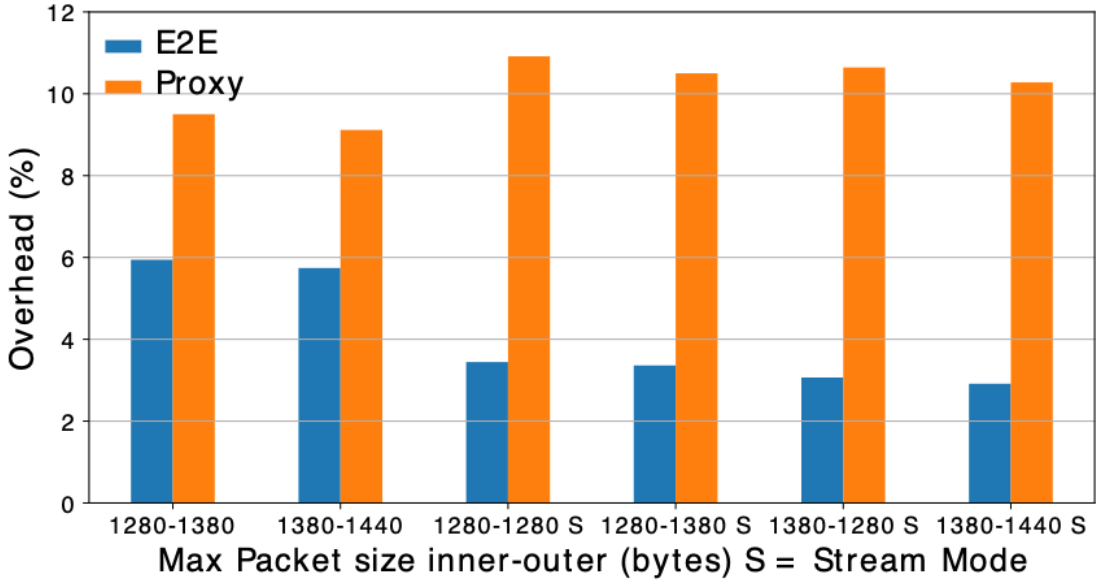
Docker-based emulation setup

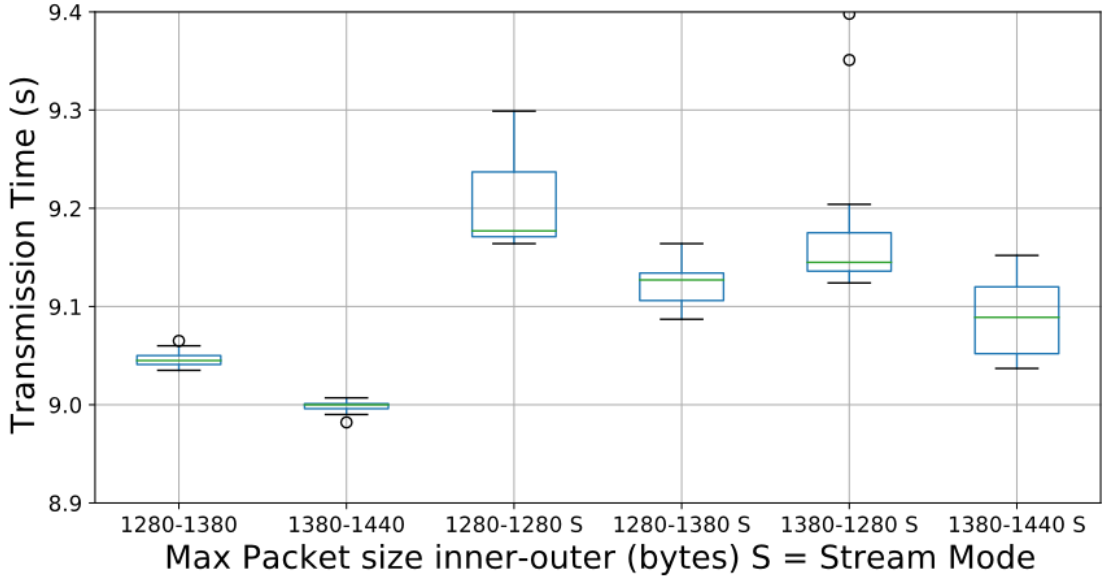
- Use of `traffic control (tc)` to emulate network condition



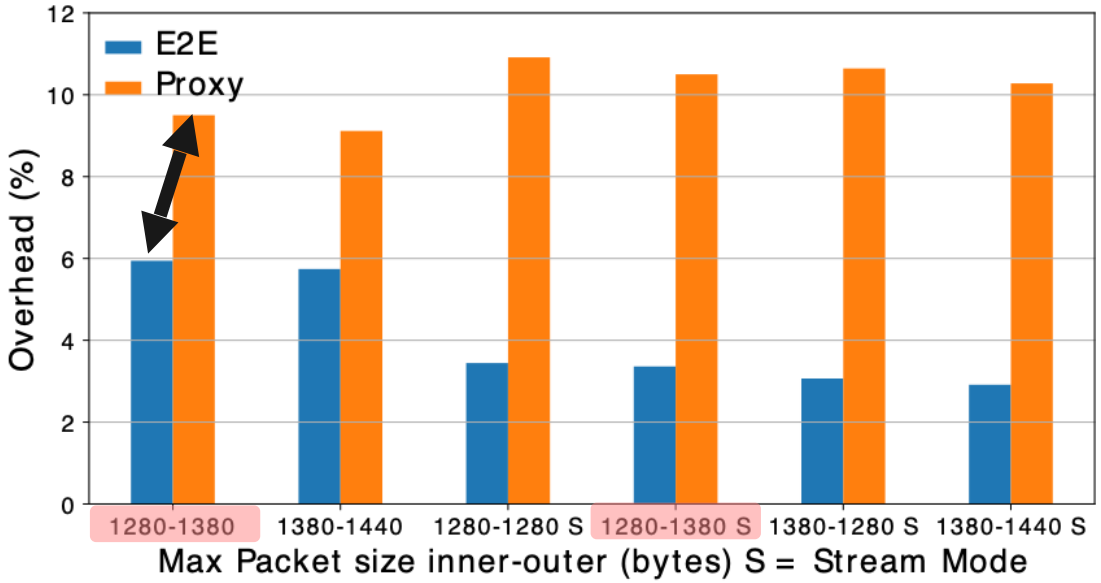
- Link between the proxy and server: **no bandwidth limitation; fixed one-way delay of 25 ms**
- Baseline configuration of the link between the client and proxy: **10 Mbit/s; 5 ms; buffer depth 200 ms**

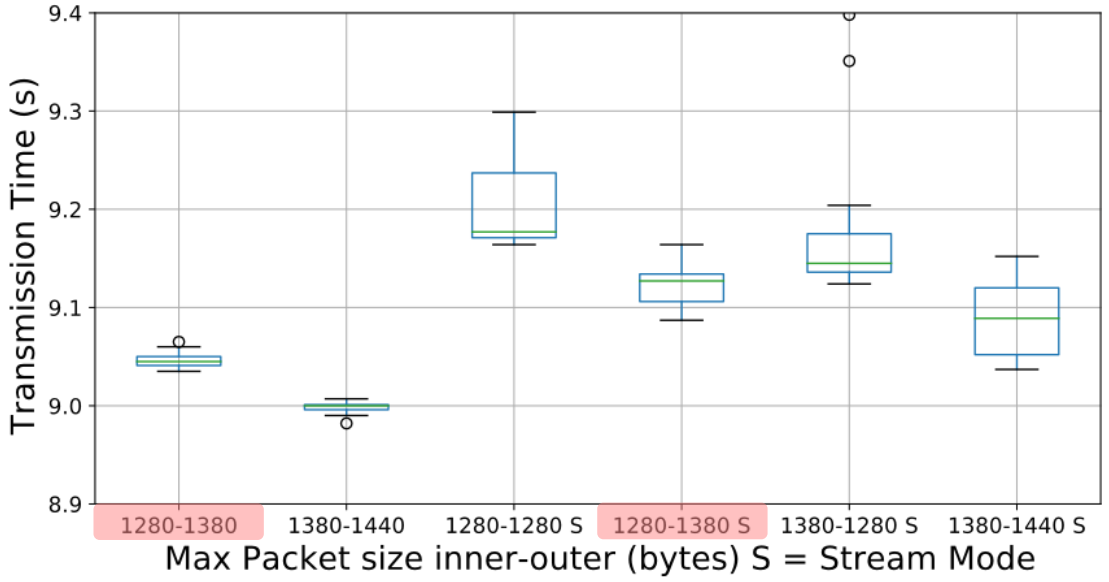
Bit overheads and MTU



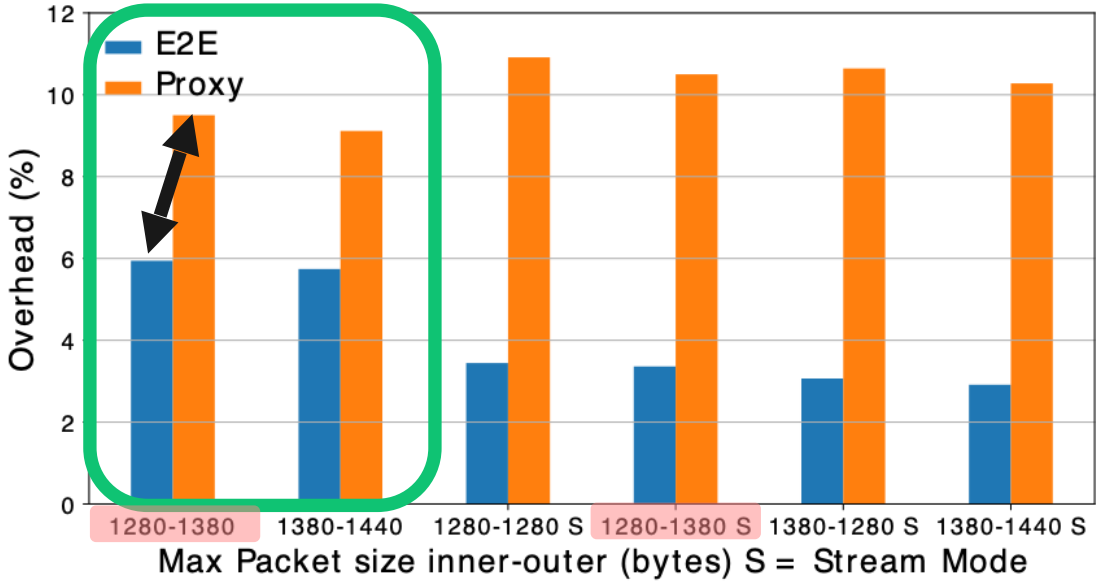


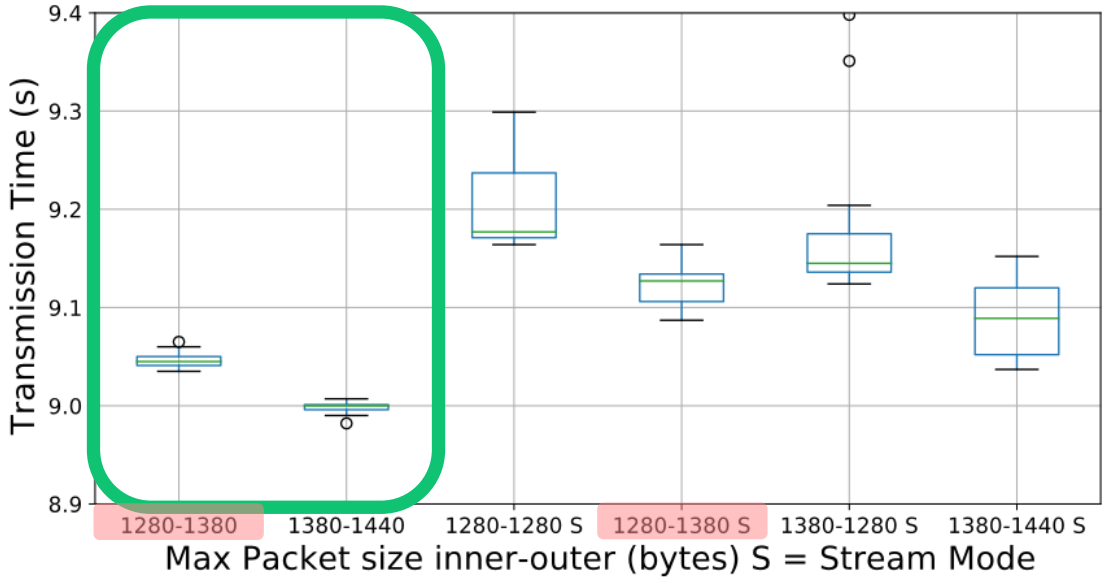
Bit overheads and MTU





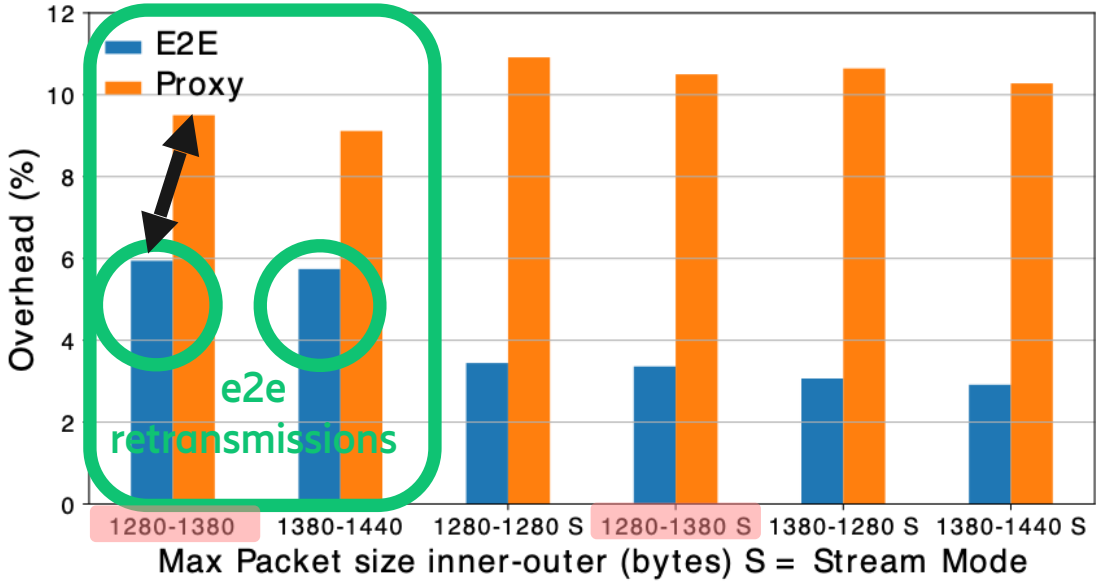
Bit overheads and MTU

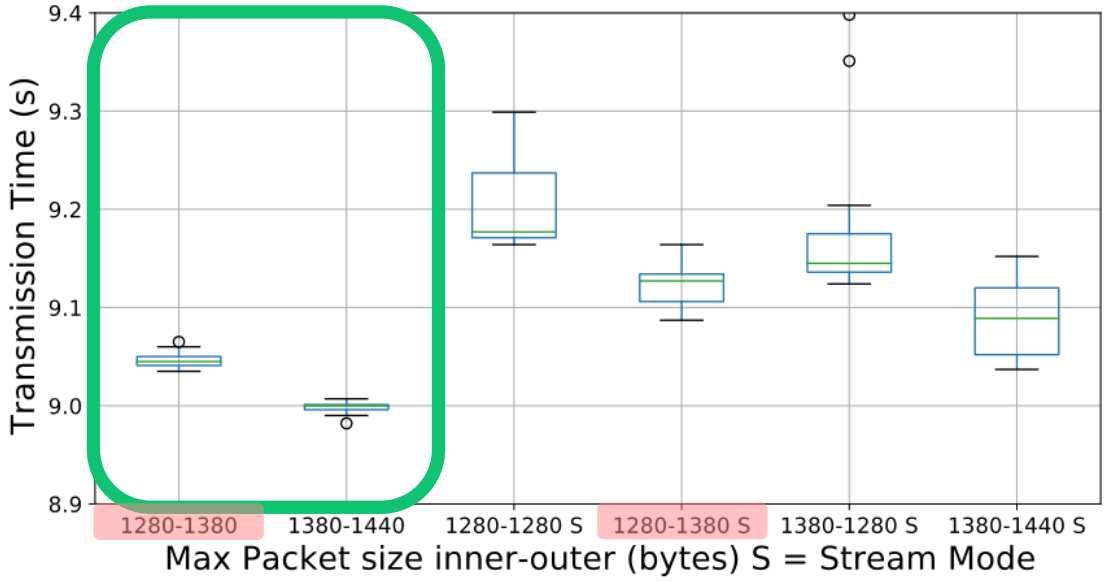




- In datagram mode the tunnel MTU must be larger than the end-to-end MTU

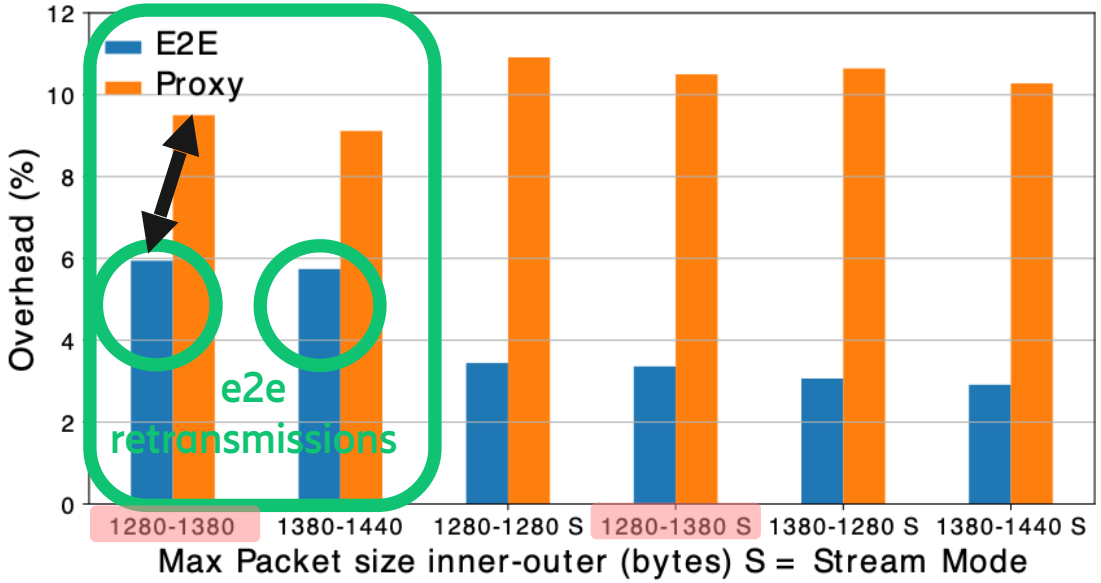
Bit overheads and MTU

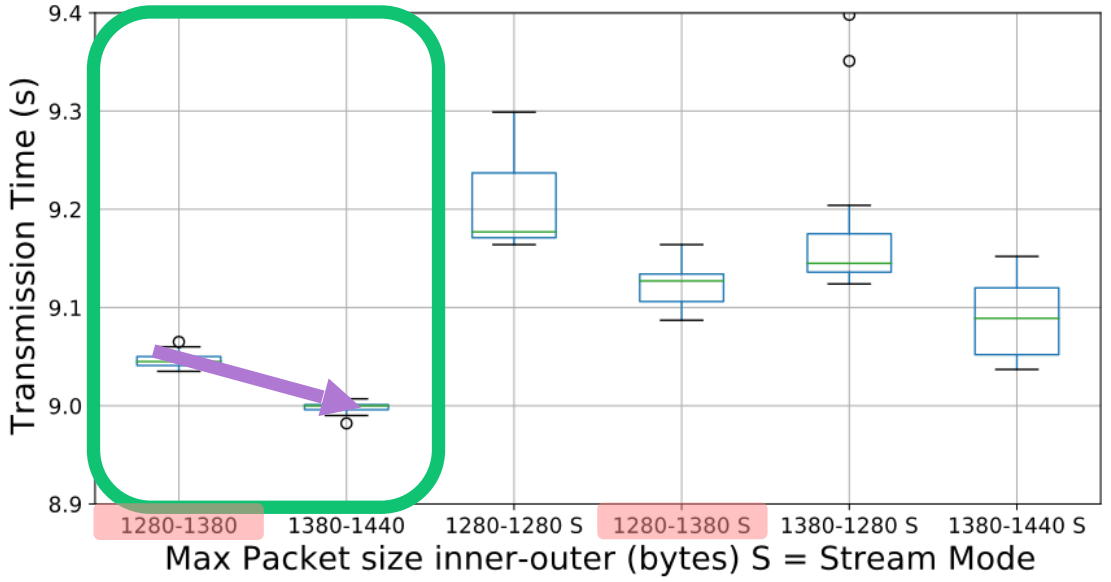




- In datagram mode the tunnel MTU must be larger than the end-to-end MTU

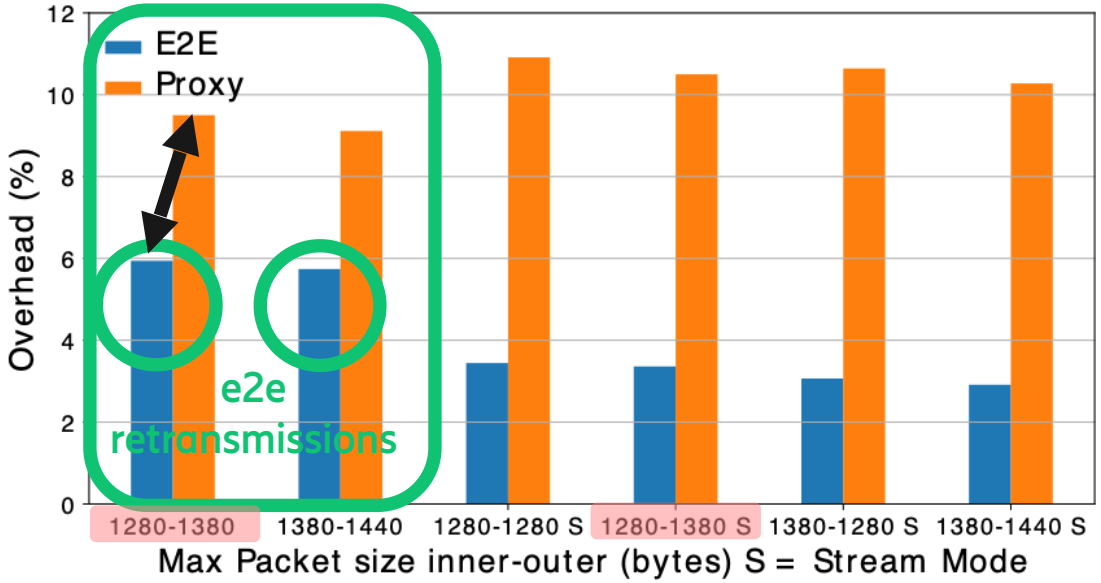
Bit overheads and MTU

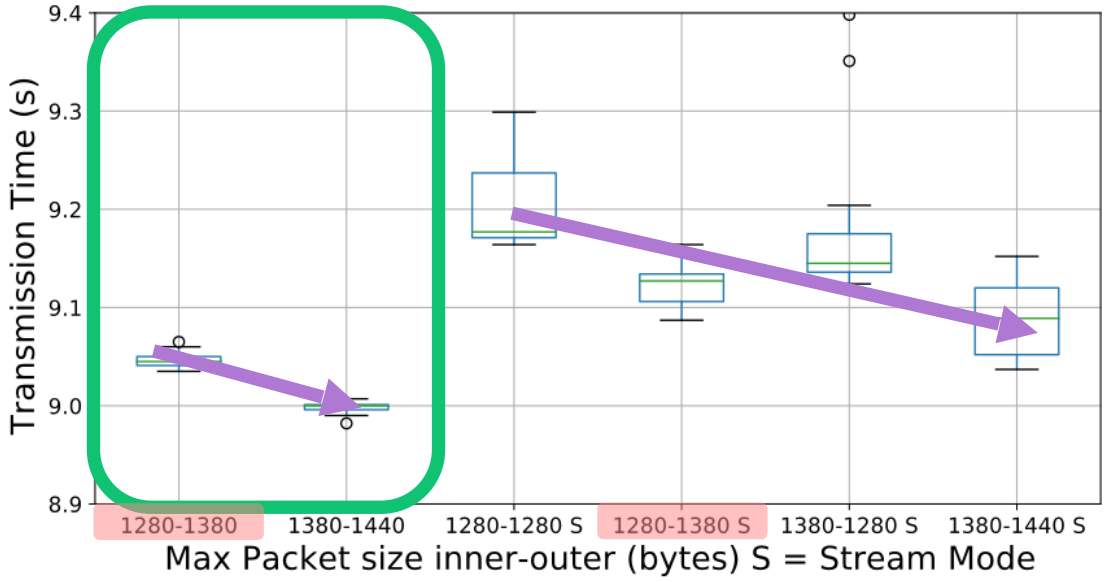




- In datagram mode the tunnel MTU must be larger than the end-to-end MTU
- Datagram mode has end-to-end retransmissions but is still less overhead under normal conditions

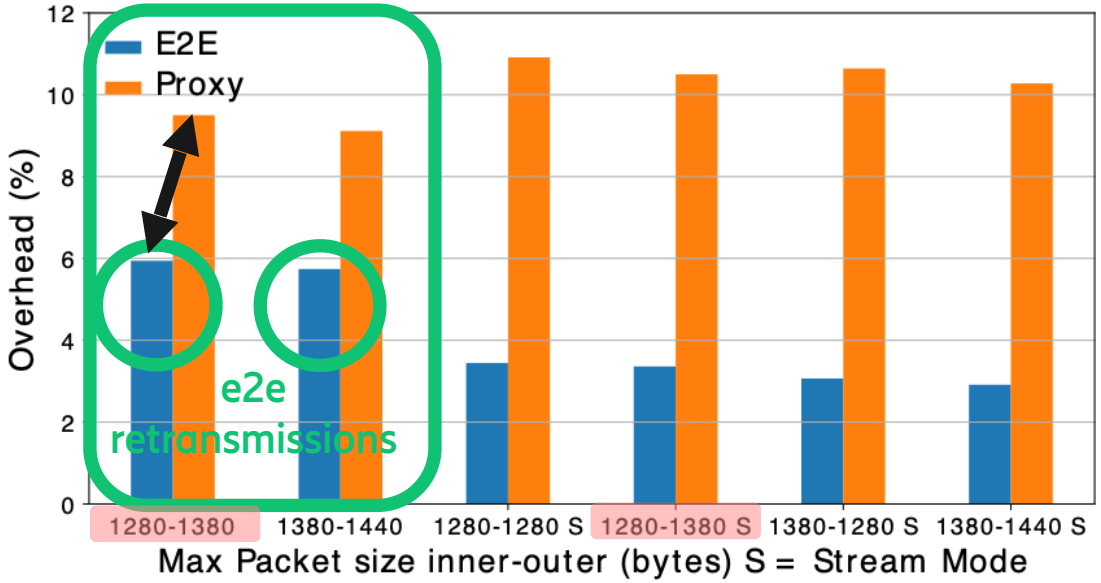
Bit overheads and MTU

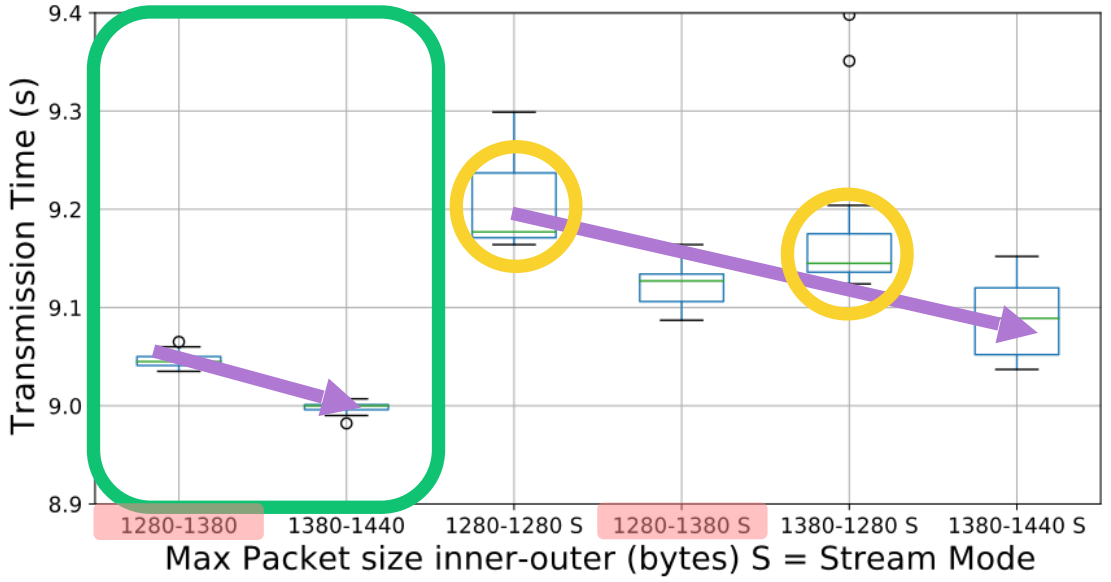




- In datagram mode the tunnel MTU must be larger than the end-to-end MTU
- Datagram mode has end-to-end retransmissions but is still less overhead under normal conditions
- As expected, larger MTUs reduce overhead and are faster

Bit overheads and MTU

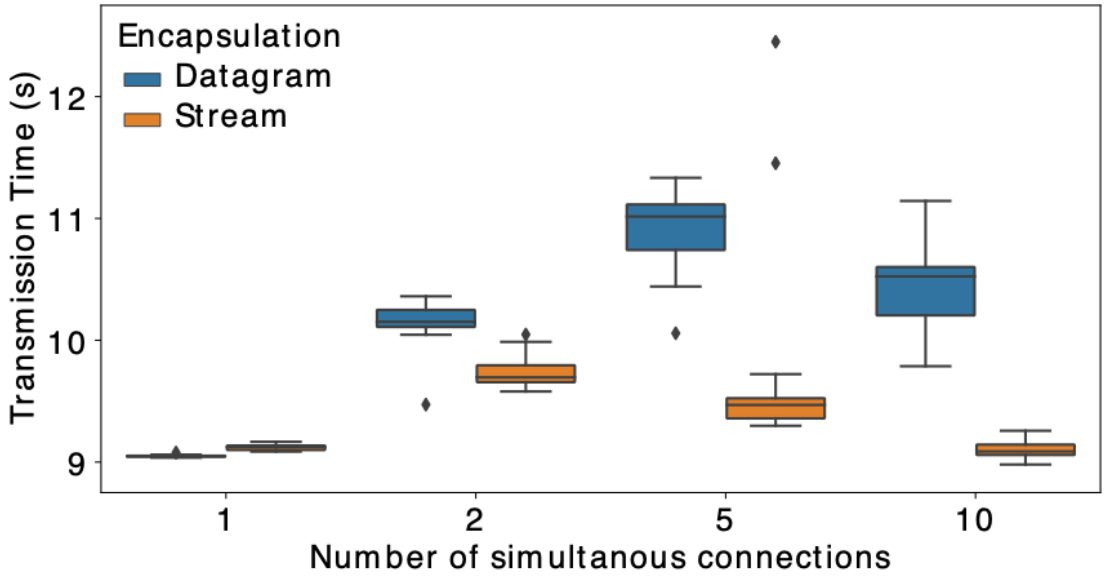


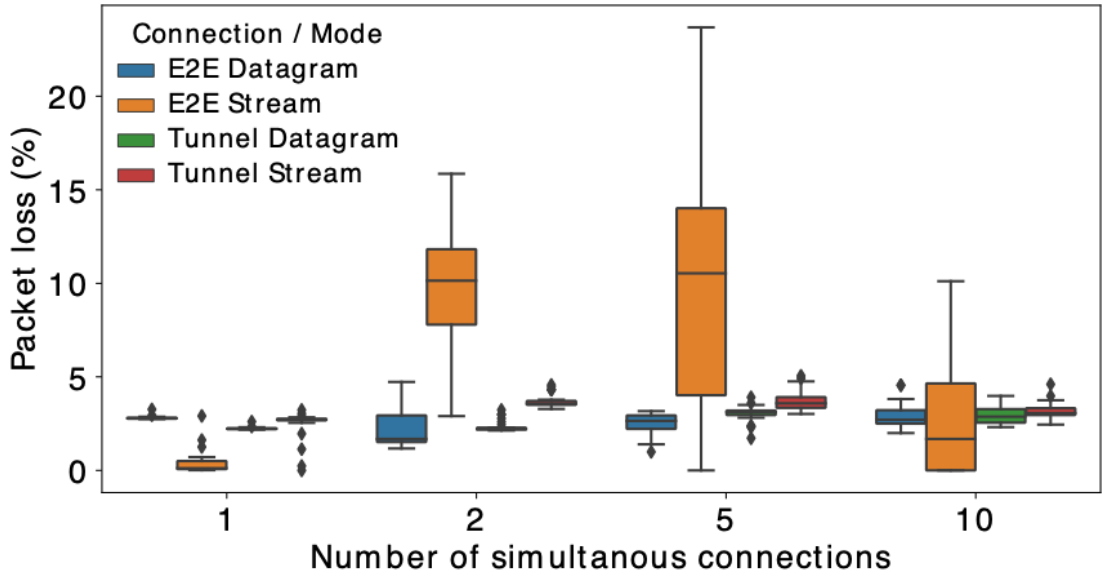


- In datagram mode the tunnel MTU must be larger than the end-to-end MTU
- Datagram mode has end-to-end retransmissions but is still less overhead under normal conditions
- As expected, larger MTUs reduce overhead and are faster
- Degradation in performance when e2e packets are split over multiple tunnel packets

Using multiple parallel CONNECT-UDP requests

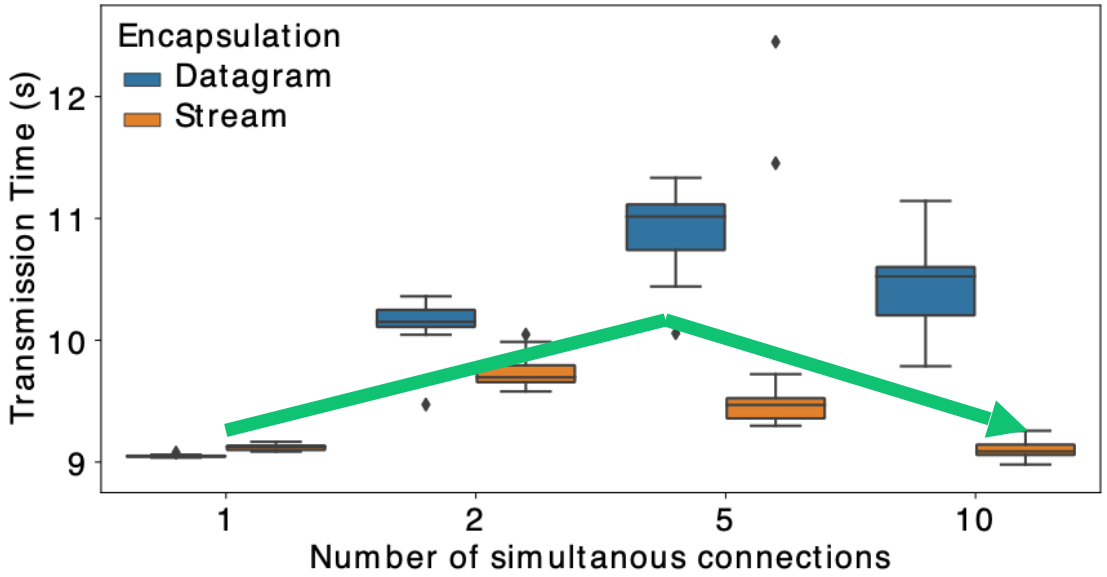
- 10 MB of divided equally over multiple e2e connections & multiplexed over a single tunnel connection

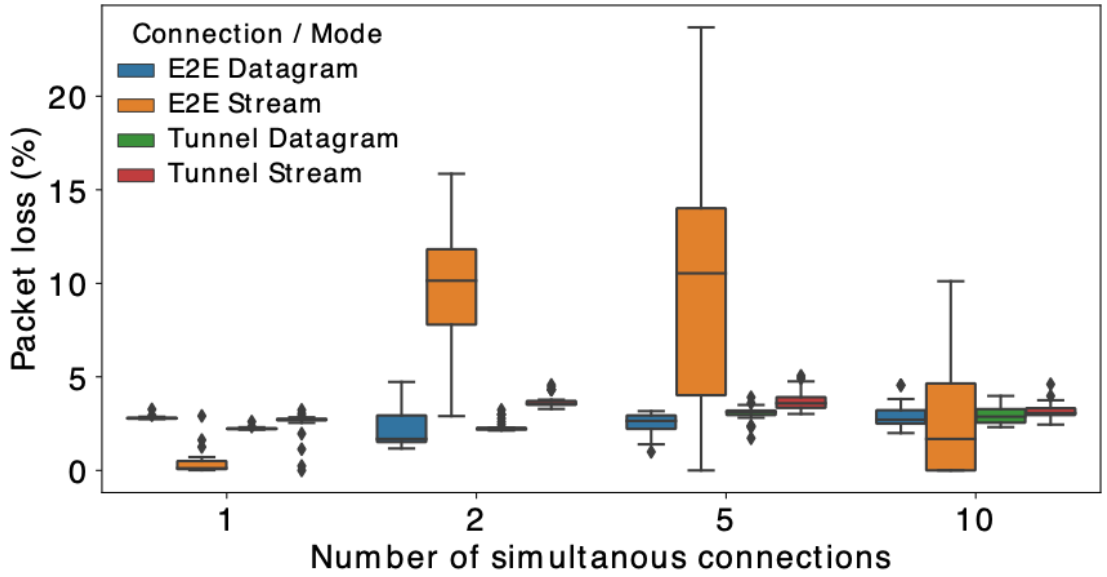




Using multiple parallel CONNECT-UDP requests

- 10 MB of divided equally over multiple e2e connections & multiplexed over a single tunnel connection

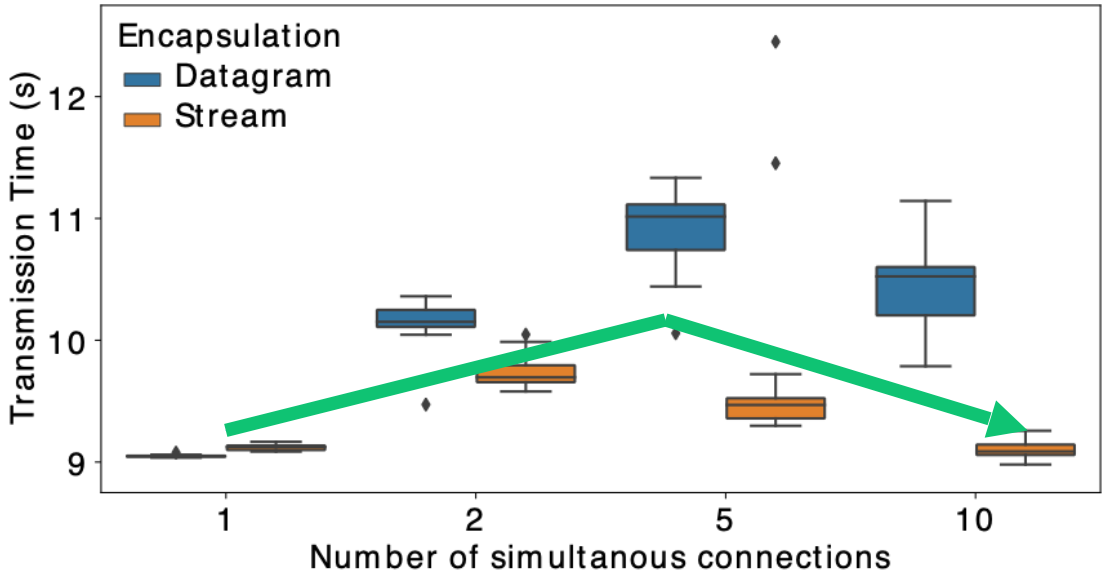


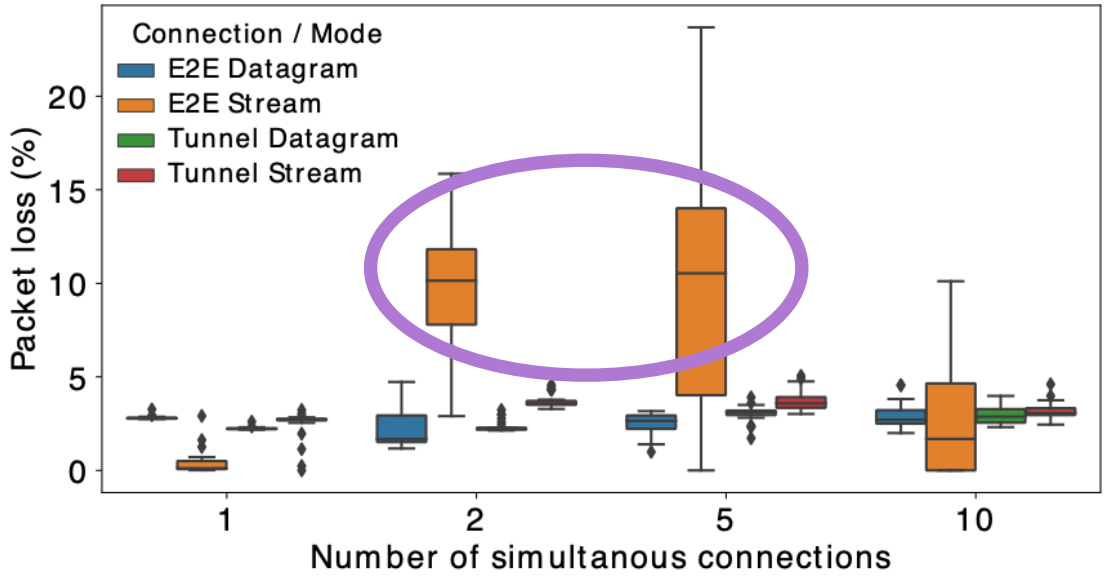


- Slightly increased overhead due to connection establishment and CONNECT-UDP request

Using multiple parallel CONNECT-UDP requests

- 10 MB of divided equally over multiple e2e connections & multiplexed over a single tunnel connection

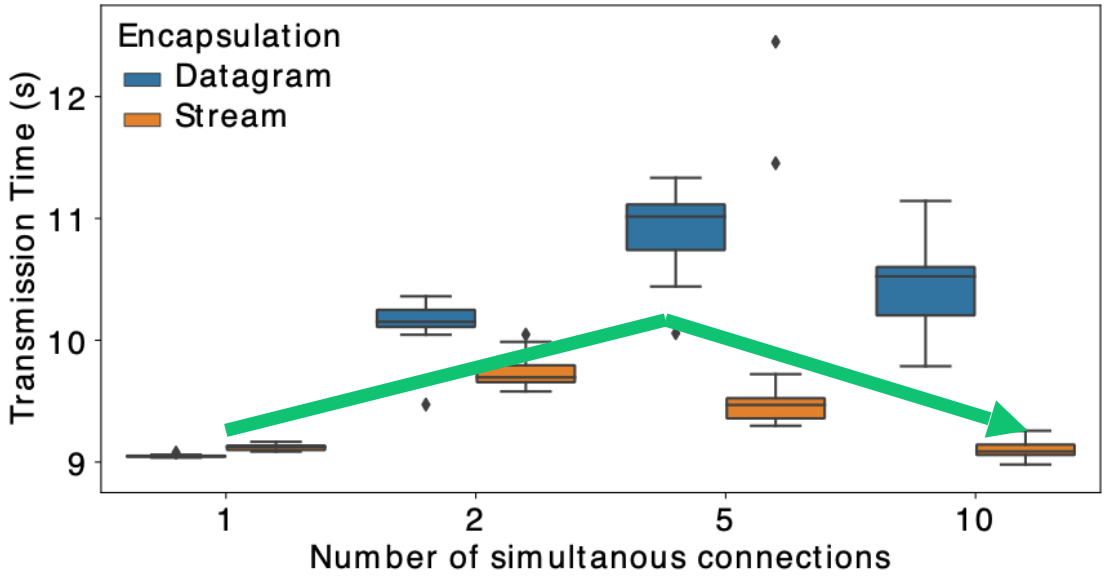


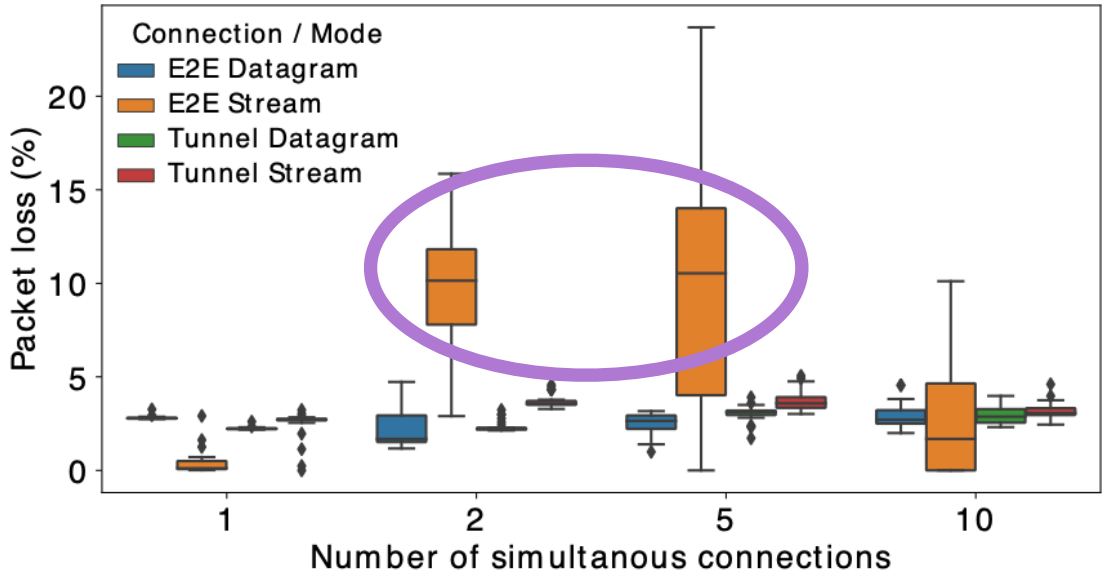


- Slightly increased overhead due to connection establishment and CONNECT-UDP request
- In stream mode: Increased packet loss caused by the stream scheduling in aioquic

Using multiple parallel CONNECT-UDP requests

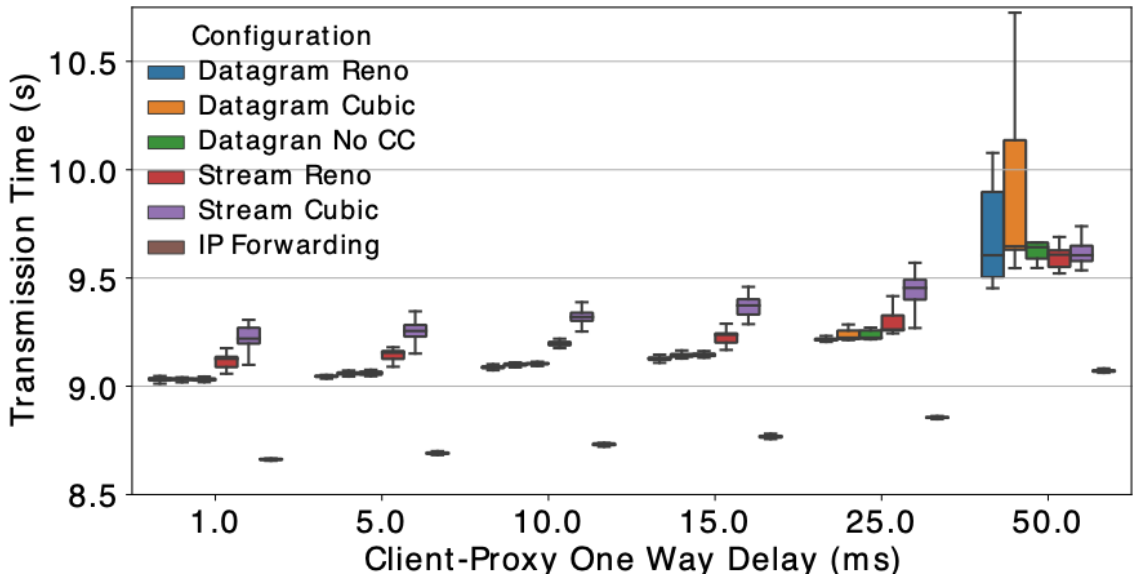
- 10 MB of divided equally over multiple e2e connections & multiplexed over a single tunnel connection

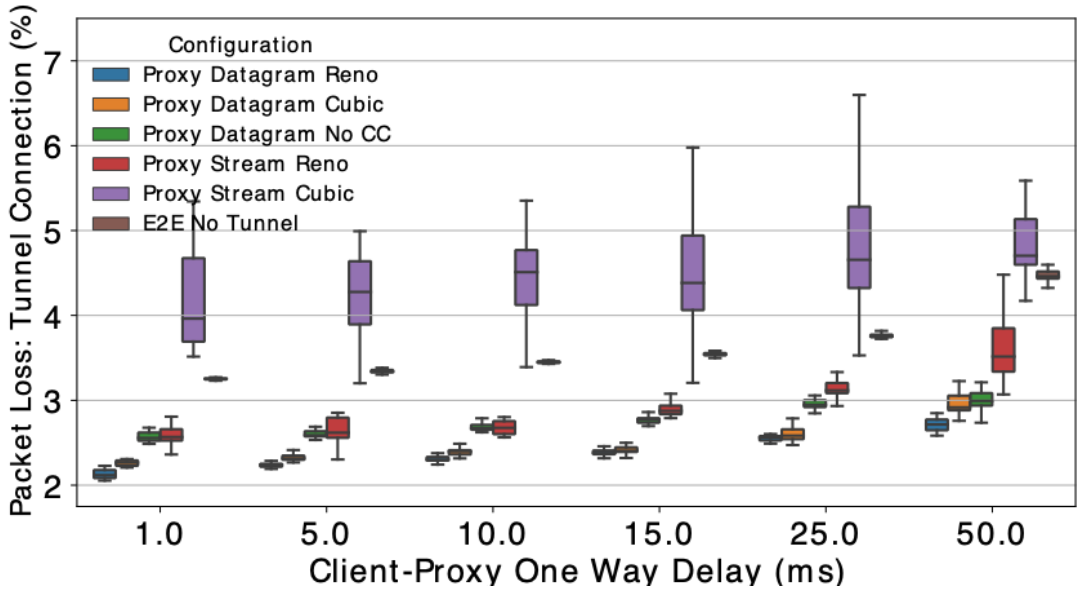




- Slightly increased overhead due to connection establishment and CONNECT-UDP request
- In stream mode: Increased packet loss caused by the stream scheduling in aioquic
- In datagram mode: Early congestion losses of competing flows slow down transmission and increase in the number of end-to-end packets with 1.7, 5.3, and 8.1 % respectively

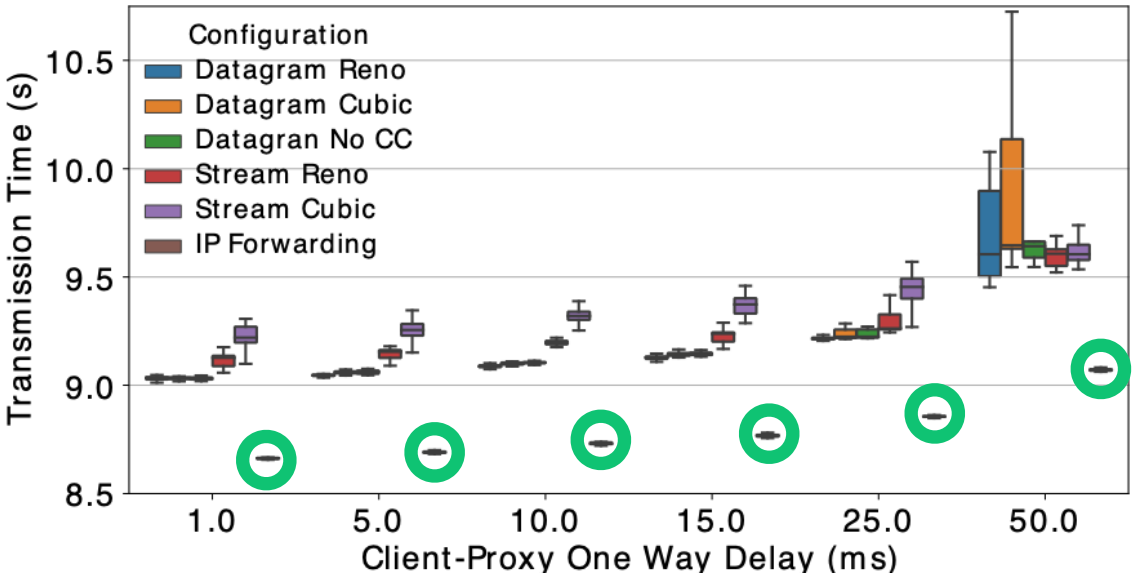
Impact of delays and congestion control

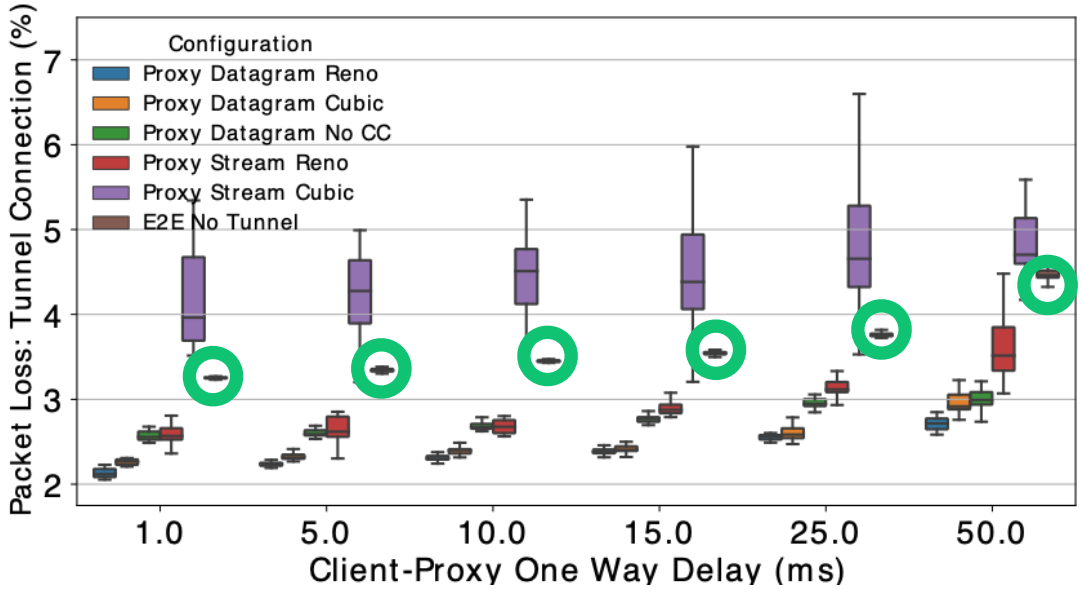




Impact of delays and congestion control

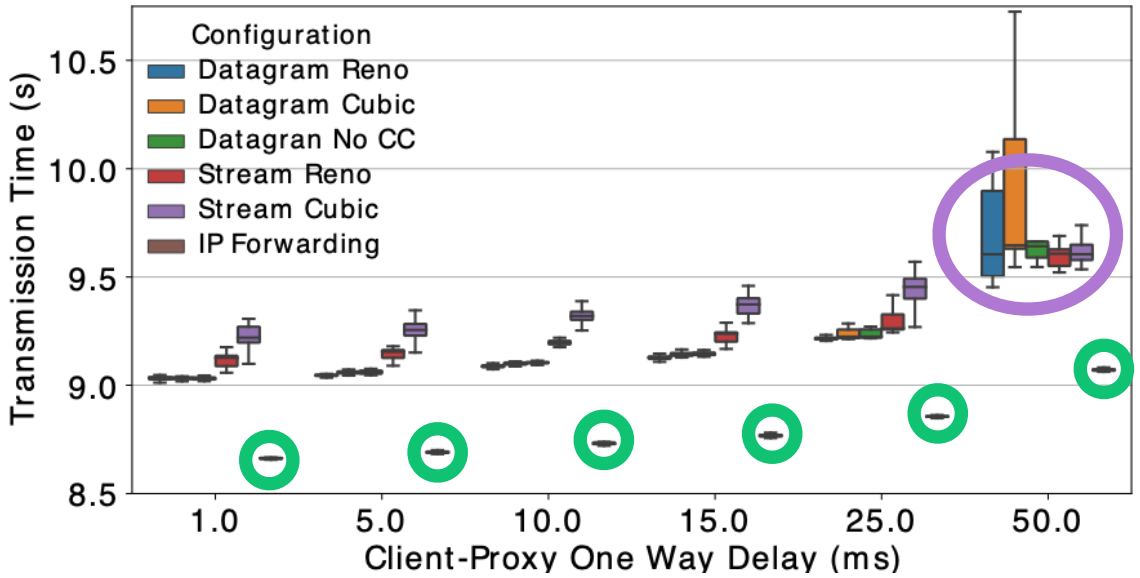
- Lower transmission time despite high e2e losses for non-proxied connection
  - For datagram mode an overhead of approximately 3% corresponds to at least 240ms at a bottleneck speed of 10 Mbps for a 10 MB message
  - Further, large buffers at the proxy increase the total transmission time despite lower losses

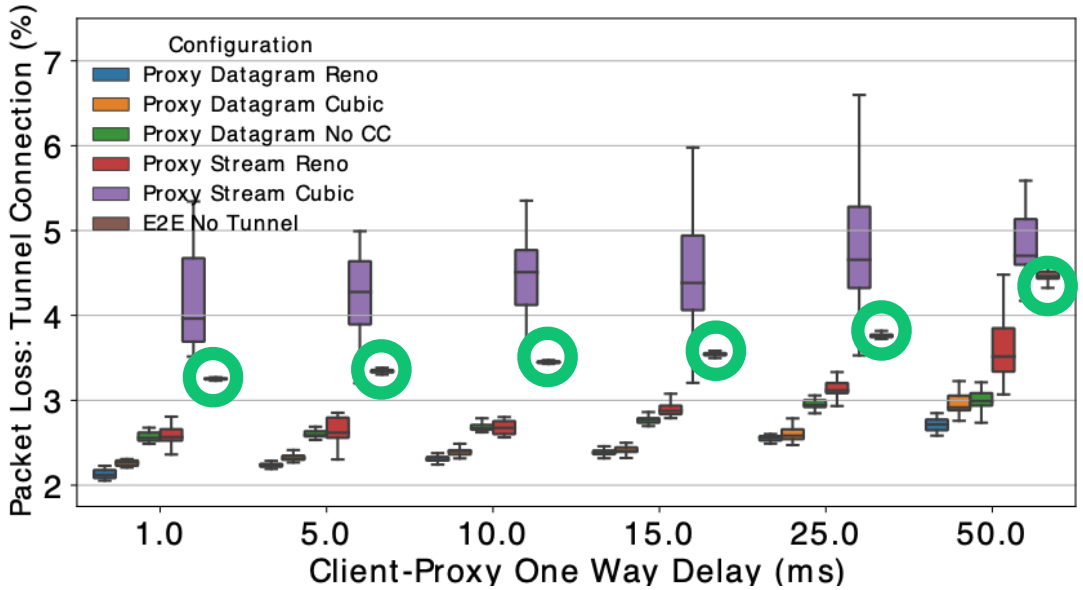




Impact of delays and congestion control

- Lower transmission time despite high e2e losses for non-proxied connection
  - For datagram mode an overhead of approximately 3% corresponds to at least 240ms at a bottleneck speed of 10 Mbps for a 10 MB message
  - Further, large buffers at the proxy increase the total transmission time despite lower losses

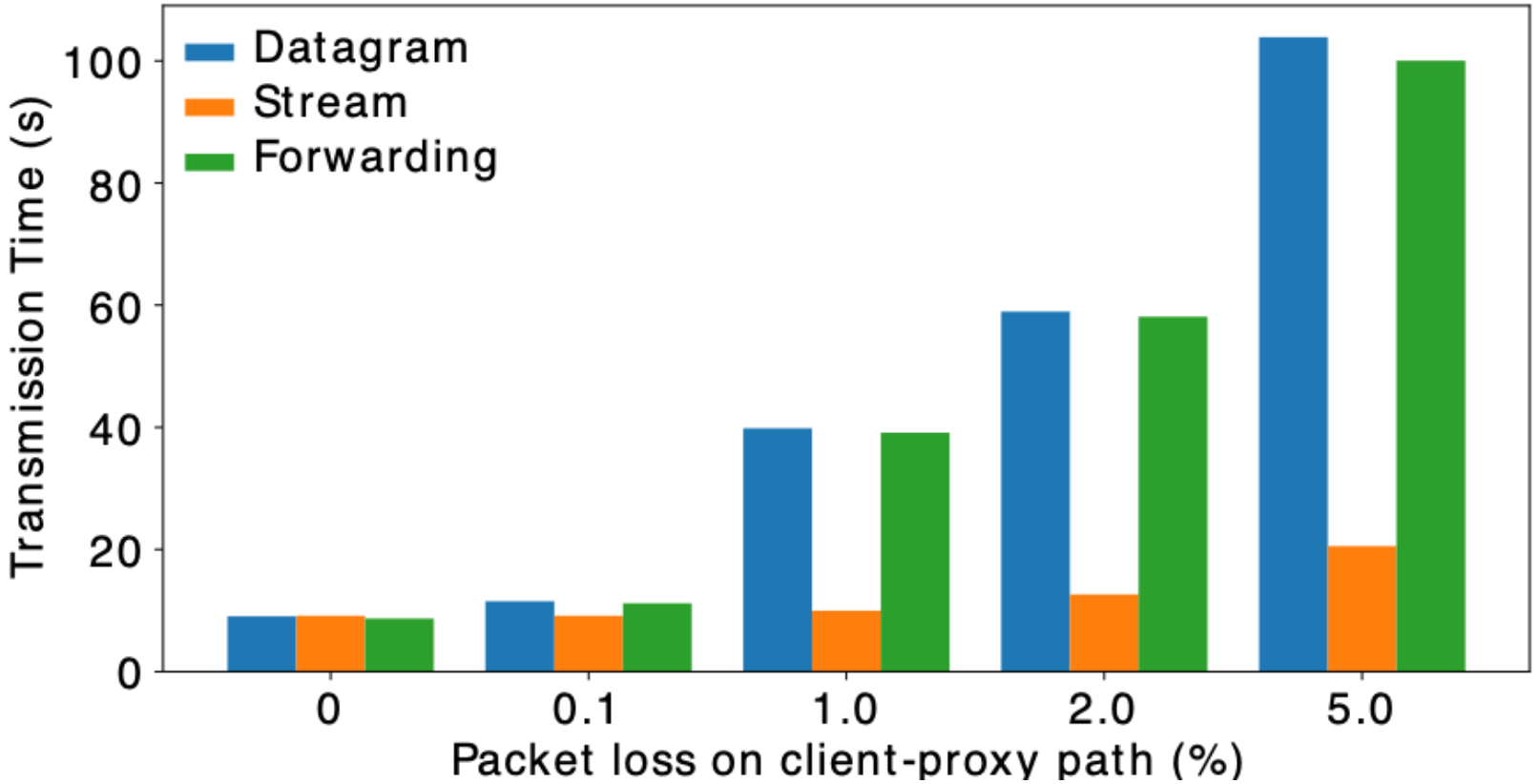




- Impact of nested congestion control, especially in datagram mode, needs further investigations

Using stream-mode for local loss recovery

- In stream mode transmission time is improved significantly, especially for high loss rates



Summary and conclusion

- MASQUE implementation based on the `aioquic` QUIC and HTTP/3 stack and docker-based emulation environment to test and evaluate various basic network scenarios
- Assessment of tunnel overhead and interactions between the end-to-end and tunnel connections
  - e.g. the stream scheduling used by the proxy when sending data reliably can have significant performance impacts
  - Reliable delivery tends to hide congestion signals from the target server, leading to both inflated RTTs and high resource consumption at the proxy
  - Insights of this proxy-based setup are limited by potential inefficiencies in the underlying stack and the simple topology of the emulation setup
- Use of stream mode can provide an opportunity to utilize mobile network configurations with simpler link layer loss recovery schemes and only use local loss recovery when explicitly required by the client