Sustained Throughput Performance of QUIC Implementations

Michael König*, Oliver P. Waldhorst‡, Martina Zitterbart*

*Institute of Telematics, Karlsruhe Institute of Technology (KIT), {m.koenig, martina.zitterbart, roland.bless}@kit.edu
‡Institute of Applied Research, Karlsruhe University of Applied Sciences (H-KA), oliver.waldhorst@h-ka.de
Motivation

"QUIC is a secure general-purpose transport protocol." [RFC9000]

Our research indicated slow throughput performance:
A QUIC-based prototype achieved ~200 Mbit/s
on a 10 Gbit/s capable testbed…

Related work

- Primarily focused on latencies and flow completion times
- Only few prior evaluations on sustained throughput in high bandwidth environments
Evaluation Setup

Setup Sender, SW-Switch, Receiver:
- CPU: Intel Xeon W-2145, 3.7–4.5 GHz, 8 Cores
- RAM: 128 GB (4x 32 GB DDR4 with 2666 MT/s)
- NIC: Intel X550-T2 (10 Gbit/s)
- OS: Linux Ubuntu 22.04.1 LTS, Kernel 5.15.0-56

- Emulation of Delay, Bandwidth, Loss
Evaluated Implementations

Six popular QUIC implementations with traffic generators (perf clients) available
- lsquic (Litespeed)
- msquic (Microsoft)
- mvfst (Facebook)
- s2n-quic (Amazon)
- picoquic
- quinn

TCP and (pure) UDP as comparison
- iperf3
- netperf

(For all TCP and QUIC traffic: Cubic as congestion control algorithm)
Results: Sustained Throughput

Average throughput of one single flow (10 runs, each 30s)

UDP 9.74 Gbit/s
Results: Sustained Throughput

Average throughput of one single flow (10 runs, each 30s)
Results: Sustained Throughput

Average throughput of one single flow (10 runs, each 30s)
Results: Sustained Throughput

Average throughput of one single flow (10 runs, each 30s)

UDP data path through the Linux Kernel is no bottleneck for QUIC
Results: Sustained Throughput

TCP* significantly outperforms QUIC implementations (from 16.1 % up to 297.5 %)

*TCP limited by testbed – Single TCP flow can achieve even 40+ Gbit/s [2]

UDP data path through the Linux Kernel is no bottleneck for QUIC
Potential Reasons for Limitations

msquic

Limited by single core performance (no multi-threading)
Potential Reasons for Limitations

**msquic**

Limited by single core performance (no multi-threading)

**lsquic**

Scheduling between CPU cores degrades throughput
Potential Reasons for Limitations

**msquic**

Limited by single core performance (no multi-threading)

**lsquic**

Scheduling between CPU cores degrades throughput

→ Inefficient Usage of CPU Resources
Impact of Cryptography

→ QUIC's performance gap: More than overhead by cryptography
Evolution of QUIC Throughput Performance

QUIC Implementations already getting quicker

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Picoquic</td>
<td>489 Mbit/s</td>
<td>2.68 Gbit/s</td>
<td>5.48x</td>
</tr>
<tr>
<td>Mvfst</td>
<td>325 Mbit/s</td>
<td>2.40 Gbit/s</td>
<td>7.38x</td>
</tr>
</tbody>
</table>

Throughput Comparison with [3] from 2020
Further Issues

*Packet Loss*

- netperf TCP
- iperf3 TCP
- Isquic
- msquic
- quinn
- s2n-quic

*Packet Reordering*

- netperf TCP
- iperf3 TCP
- Isquic
- msquic
- quinn
- s2n-quic

→ QUIC implementations stronger affected by packet losses than TCP

→ mvfst, quinn, Isquic, and s2n-quic misinterpret reordered packets as losses
ACK Ratios

![Bar chart showing ACK ratios for different QUIC variants](chart.png)

- Isquic: 23.46
- Msquic: 72.18
- Mvfst: 20.02
- Picoquic: 104.34
- Quinn: 49.71
- S2n-quic: 23.27
ACK Ratios

→ ACK Ratio seemingly not correlated with throughput performance
Impact of Offloading

Offloading can improve performance
Conclusion

- Current QUIC implementations: Not up to par with TCP regarding sustained throughput rates
  - QUIC's performance gap: More than overhead by cryptography
  - Inefficient usage of CPU resources

- Possible solutions
  - Better usage of multiple CPU cores
  - Avoid scheduling between CPU cores
  - Offloading to (optimized) Kernel functions
References


