Examining the Heterogeneous Throughput Performance Landscape of QUIC Implementations*

Michael König[§], Sebastian Rust[‡], Martina Zitterbart[§], Björn Scheuermann[‡], Presenter: Roland Bless[§]

[§] Institute of Telematics, Karlsruhe Institute of Technology, Karlsruhe, Germany, {m.koenig, martina.zitterbart, roland.bless}@kit.edu
[‡] Technical University of Darmstadt, Darmstadt, Germany, {sebastian.rust@, scheuermann@kom}.tu-darmstadt.de
* In Proceedings of IFIP Networking 2025, Limassol, Cyprus 2025

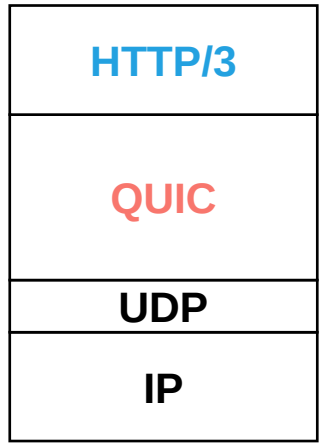
QUIC Throughput: Status Quo

TECHNISCHE UNIVERSITÄT DARMSTADT

Karlsruhe Institute of Technology

- QUIC's adoption is increasing
  - Many implementations exist (from Google, Microsoft, Amazon, …)
  - More applications: e.g., DNS-over-QUIC, Samba-over-QUIC
- Existing throughput evaluations indicate high variability between different implementations [1][2][3]
  - Include application overhead: QUIC+HTTP [1]
  - Only use QUIC-only traffic [2][3]

Each performed with different setups (hardware, implementation version, …)
→ difficult to compare



QUIC web stack

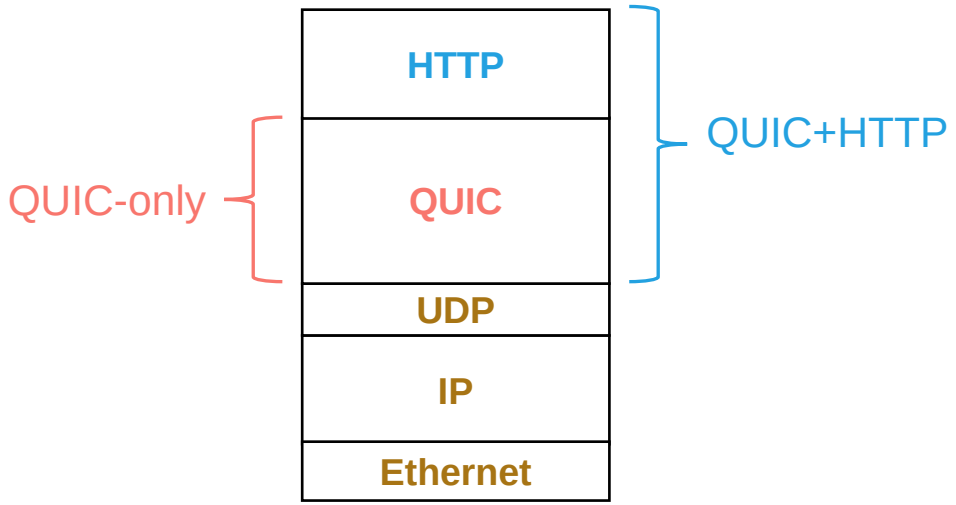
Our contributions:

→ Direct comparison of QUIC+HTTP and QUIC-only performance in same testbed
→ Additional comprehensive performance overview across the network stack

[1] Benedikt Jaeger et al. "QUIC on the highway: evaluating performance on high-rate links". IFIP Networking'23. 2023.
[2] Michael König et al. "QUIC(k) Enough in the Long Run? Sustained Throughput Performance of QUIC Implementations". LCN'23. 2023
[3] Xiangrui Yang et al. "Making QUIC Quicker With NIC Offload". EPIQ'20. 2020.

Methodology

- QUIC throughput performance across the network stack
  - Application level: QUIC+HTTP traffic
  - Transport level: QUIC-only traffic
  - Lower level: Offloading & MTU
- Identify possible performance bottlenecks (e.g., Context switches, CPU Resources, …)



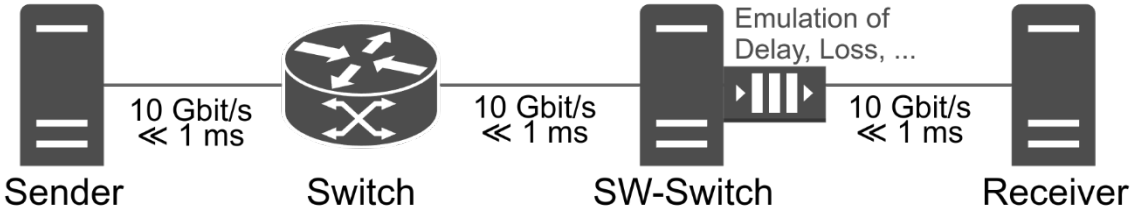
Performance Correlated?

QUIC+HTTP results

QUIC-only results

Performance Correlated?

Experiment Setup: Testbed

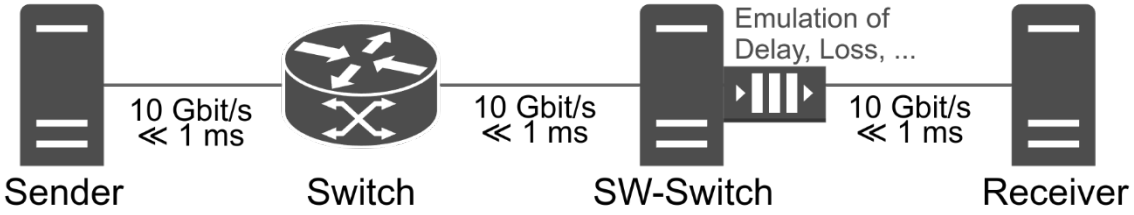


- 10 Gbit/s capable testbed
  - Sender, receiver, hardware switch, SW-switch
  - Hard- and software identical to [2]
  - Common network performance tuning applied [3][4]

CPU	Intel Xeon W-2145, 3.7–4.5 GHz, 8 Cores / 16 Threads
RAM	128 GB (4x 32 GB DDR4 with 2666 MT/s)
NIC	Intel X550-T2
OS	Linux Ubuntu 22.04.1 LTS
Kernel	5.15.0-56-generic

[2] Michael König et al. “QUIC(k) Enough in the Long Run? Sustained Throughput Performance of QUIC Implementations”. In: LCN’23. 2023
[3] Mario Hock et al. “TCP at 100 Gbit/s – Tuning, Limitations, Congestion Control”. In: IEEE LCN. 2019.
[4] Kevin Corre. Framework for QUIC Throughput Testing. Internet-Draft. 2021.

Experiment Setup: Testbed



- 10 Gbit/s capable testbed
  - Sender, receiver, hardware switch, SW-switch
  - Hard- and software identical to [2]
  - Common network performance tuning applied [3][4]

CPU	Intel Xeon W-2145, 3.7–4.5 GHz, 8 Cores / 16 Threads
RAM	128 GB (4x 32 GB DDR4 with 2666 MT/s)
NIC	Intel X550-T2
OS	Linux Ubuntu 22.04.1 LTS
Kernel	5.15.0-56-generic

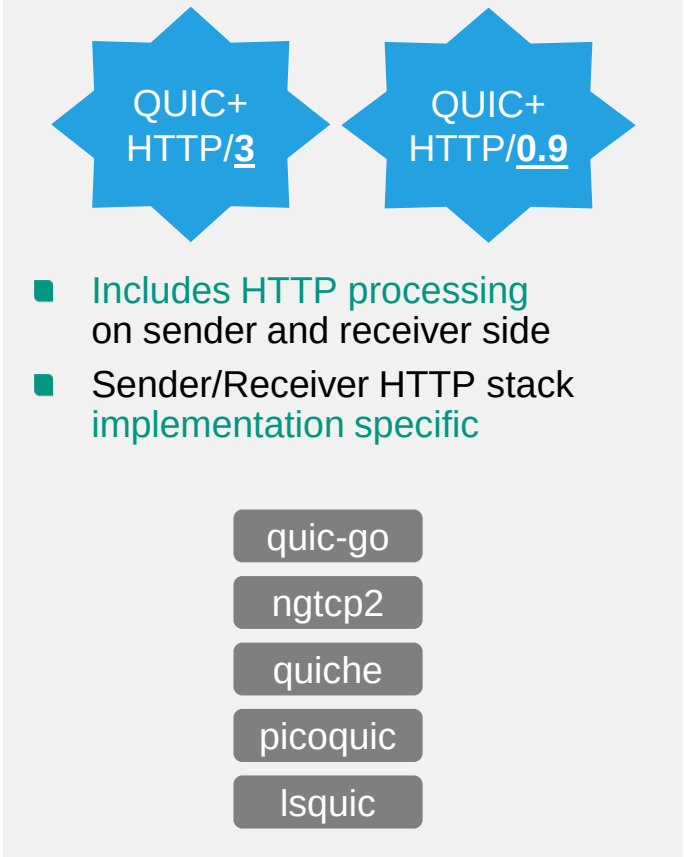
- 5 Open-source QUIC implementations
  - Popular according to GitHub stars, interactions, ...
  - All implemented in user-space
  - Written in different programming languages
  - All support Cubic as congestion control algorithm
    → Cubic used in all experiments

Implementation	Language
lsquic	C
picoquic	C
ngtcp2	C
quiche (Cloudfare)	Rust
quic-go	Go

[2] Michael König et al. “QUIC(k) Enough in the Long Run? Sustained Throughput Performance of QUIC Implementations”. In: LCN’23. 2023
[3] Mario Hock et al. “TCP at 100 Gbit/s – Tuning, Limitations, Congestion Control”. In: IEEE LCN. 2019.
[4] Kevin Corre. Framework for QUIC Throughput Testing. Internet-Draft. 2021.

Experiment Setup: Traffic Generation

QUIC InterOp Runner[5]: QUIC+HTTP



[5] https://github.com/quic-interop/quic-interop-runner
[6] https://github.com/victorstewart/quicperf

Experiment Setup: Traffic Generation

QUIC InterOp Runner[5]: QUIC+HTTP

QUIC+ HTTP/**3**

QUIC+ HTTP/**0.9**

- Includes HTTP processing on sender and receiver side
- Sender/Receiver HTTP stack implementation specific

quic-go
ngtcp2
quiche
picoquic
lsquic

QUIC-only Traffic Generators

QUIC-only: **Integrated**

- Plain QUIC packets → no HTTP stack
- Implementation specific
  - picoquic-perf for picoquic
  - lsquic-perf for lsquic

picoquic
lsquic

[5] https://github.com/quic-interop/quic-interop-runner
[6] https://github.com/victorstewart/quicperf

Experiment Setup: Traffic Generation

QUIC InterOp Runner[5]: QUIC+HTTP

QUIC+ HTTP/3

QUIC+ HTTP/0.9

- Includes HTTP processing on sender and receiver side
- Sender/Receiver HTTP stack implementation specific

quic-go
ngtcp2
quiche
picoquic
lsquic

QUIC-only Traffic Generators

QUIC-only: Integrated

- Plain QUIC packets → no HTTP stack
- Implementation specific
  - picoquic-perf for picoquic
  - lsquic-perf for lsquic

picoquic
lsquic

QUIC-only: META

- Plain QUIC packets → no HTTP stack
- Compatible with multiple implementations
- Using quicperf [6]

ngtcp2
quiche
picoquic
lsquic

[5] https://github.com/quic-interop/quic-interop-runner
[6] https://github.com/victorstewart/quicperf

Differences in Combination/Pairing

Scenario (via QUIC InterOp Runner): 1 HTTP/3 Request → 8 GiB response

Receiver \ Sender	lsquic	ngtcp2	picoquic	quic-go	quiche
lsquic	2.473	2.375	2.380	1.434	2.233
ngtcp2	2.523	4.172	3.085	1.451	3.955
picoquic	1.903	1.752	1.518	1.249	1.335
quic-go	1.318	1.264	1.346	1.291	1.220
quiche	2.537	3.192	2.486	1.248	2.972

Avg. Throughput [Gbit/s]

quiche→ngtcp2: 3.955 Gbit/s
ngtcp2→quiche: 3.192 Gbit/s
Difference: 0.763 Gbit/s

⟶ Asymmetrical performance between sending directions
Fast sender != fast receiver implementation

Comparing HTTP/0.9 with HTTP/3

TECHNISCHE UNIVERSITÄT DARMSTADT

Karlsruhe Institute of Technology

QUIC+ HTTP/0.9

Scenario (via QUIC InterOp Runner): 1 HTTP/0.9 Request → 8 GiB response

Receiver \ Sender	lsquic	ngtcp2	picoquic	quic-go	quiche
lsquic	2.75	1.92	2.34	~~-0.60~~	~~-3.26~~
ngtcp2	23.76	~~1.66~~	~~-0.73~~	~~0.41~~	~~-0.155~~
picoquic	~~-1.28~~	~~-2.85~~	~~-8.97~~	~~2.39~~	~~14.98~~
quic-go	21.97	25.88	13.71	6.91	27.11
quiche	21.04	3.99	~~-0.42~~	~~0.01~~	5.16

Relative Throughput Difference
for HTTP/0.9 instead of HTTP/3 [%]

Striked values indicate differences statistically not significant (too much variance)

Up to 27.11% faster when using HTTP/**0.9** instead of HTTP/**3.0**

→ Application protocol and its implementation can significantly impact performance

Summary QUIC+HTTP Tests

- Throughput performance varies significantly
  - Across application protocols (i.e., HTTP/3 vs HTTP/0.9)
  - Across implementations
  - Across pairings
    → sender/receiver combination matters
- Number of concurrent streams can improve throughput performance

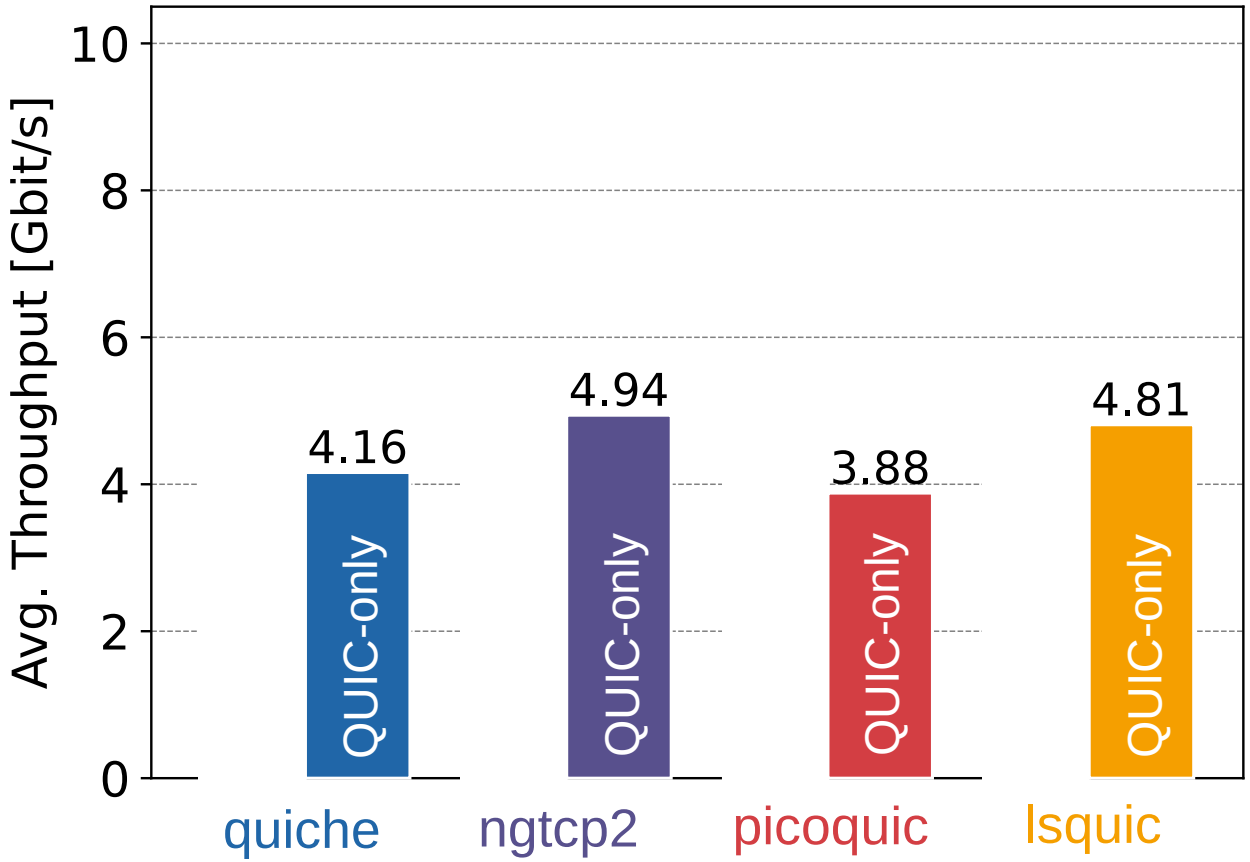
QUIC throughput performance across the network stack

Application level: QUIC+HTTP traffic ✓

Transport level: QUIC-only traffic

Lower level: Offloading & MTU

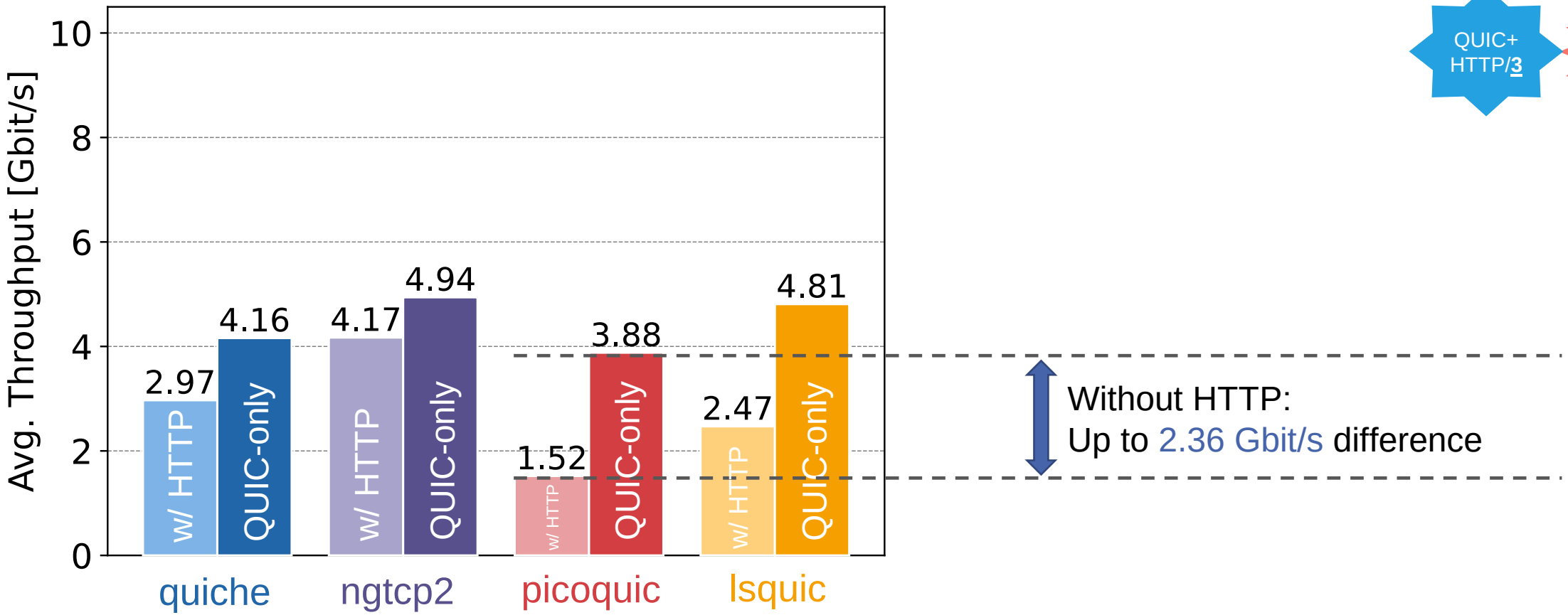
QUIC+HTTP/3 vs QUIC-only



QUIC -only

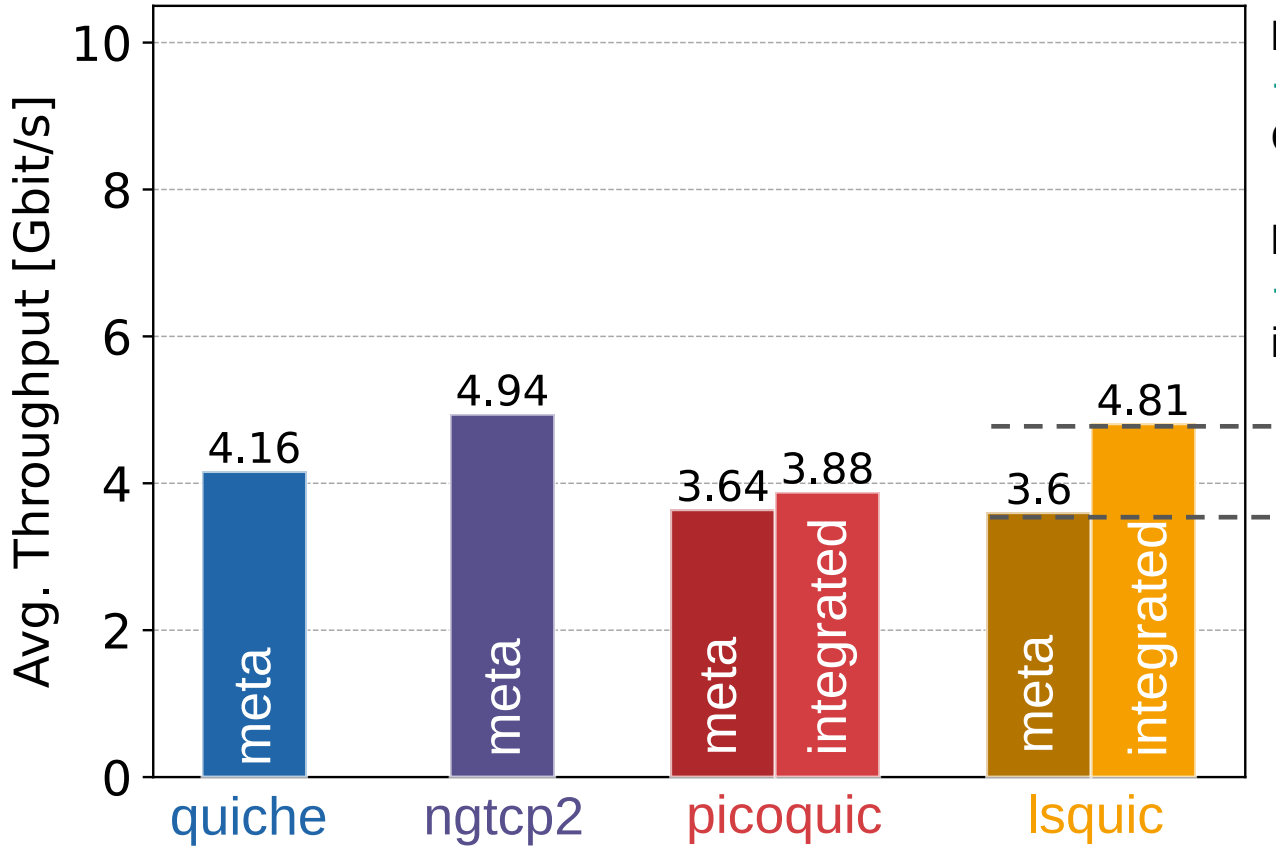
Research Group Prof. Zitterbart (KIT)
Research Group Prof. Scheuermann (TUDa)

QUIC+HTTP/3 vs QUIC-only



⟶ HTTP overhead significant &
QUIC+HTTP performance not representative for QUIC-only results (and vice versa)

Influence of Traffic Generator



Meta traffic generator
→ compatible with multiple QUIC implementations

Integrated traffic generators
→ specific for each QUIC implementations

1.21 Gbit/s difference

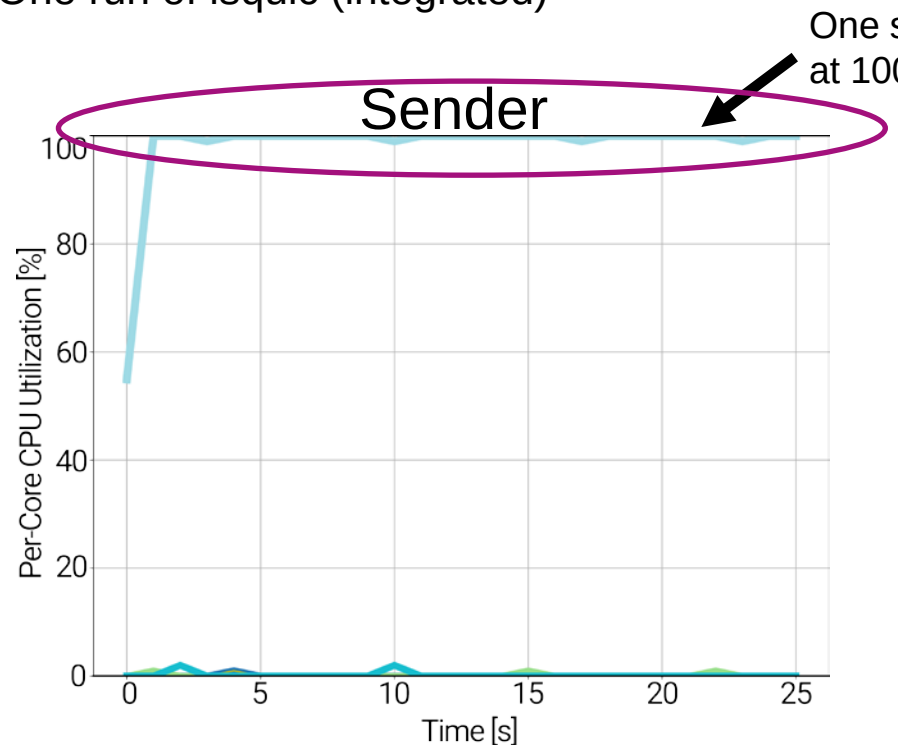
QUIC-only: **META**

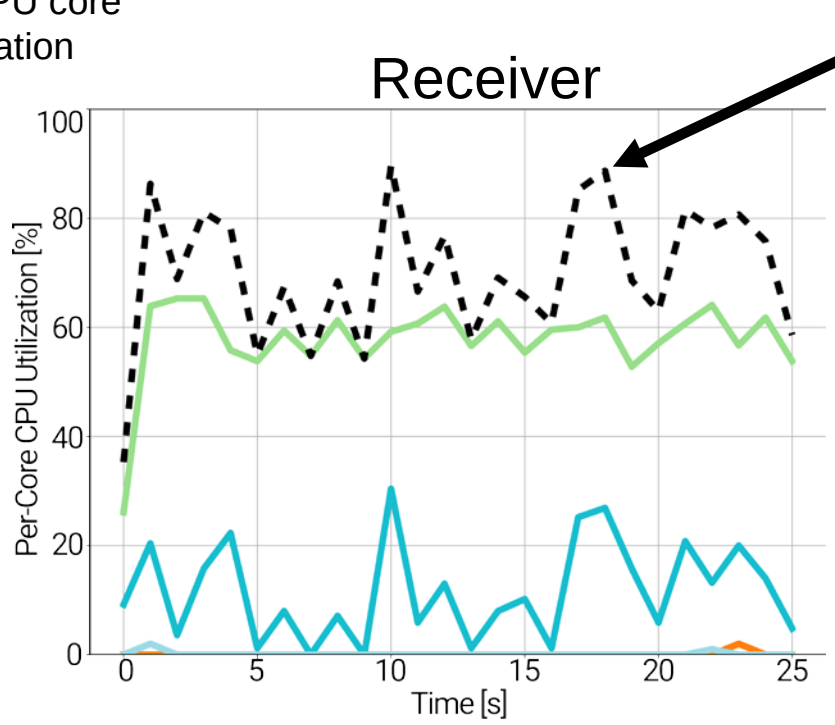
QUIC-only: **Integrated**

→ Performance of traffic generators (themselves) impact results

CPU Utilization of lsquic (integrated)

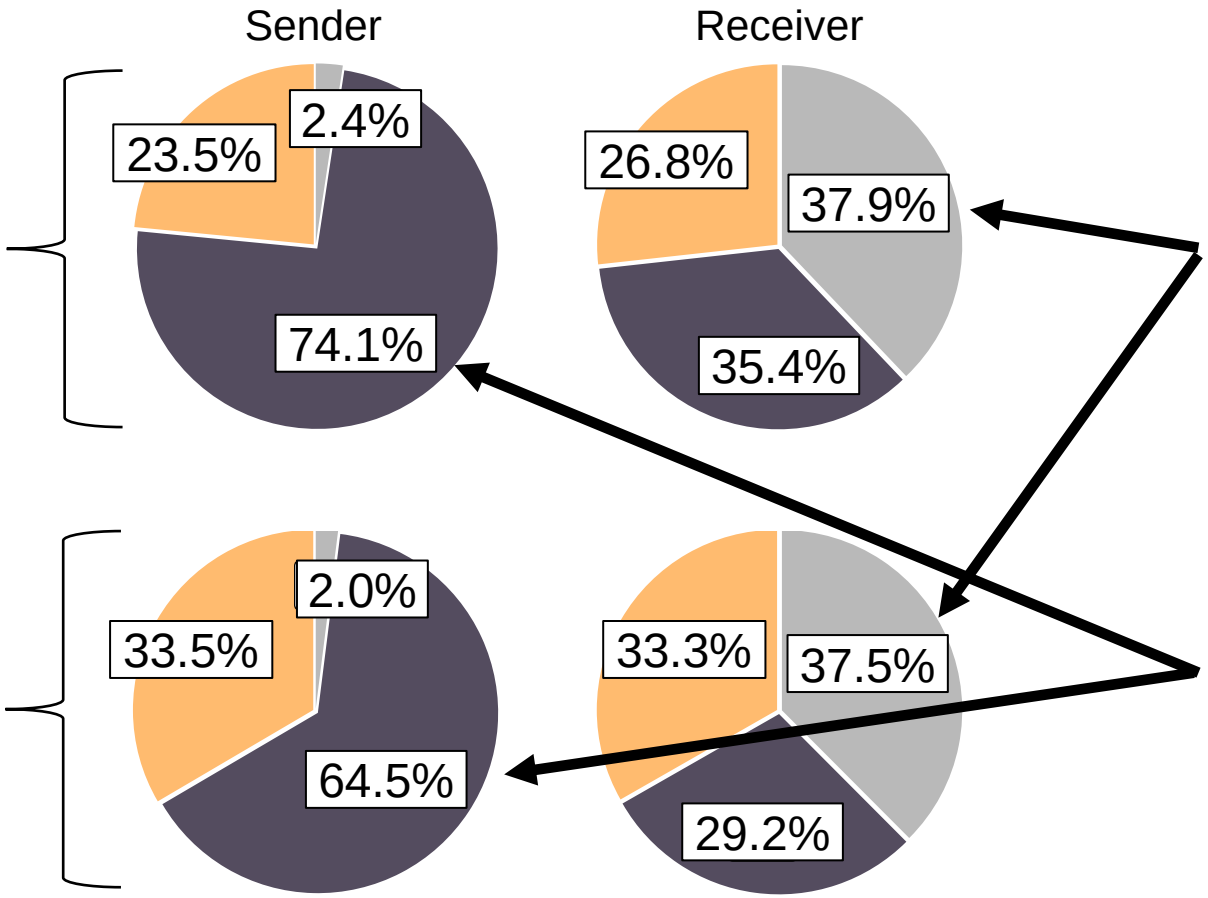
Example: One run of lsquic (integrated)





⟶ Throughput limited by single core CPU performance on sender side

CPU Time Distribution



Legend

- User
- Sys
- Idle

For both traffic generators:
Receivers up to ~**38% idle time (doing nothing)**
While senders fully occupied
→ **Limited by sender side**

Difference in kernel/sys time:
Slower(!) meta traffic generator:
→ **More syscalls/interrupts**
By more frequent `mallocs()` (inefficient!)

⟶ Performance can suffer from inefficient "application ↔ QUIC implementation" interactions

Summary QUIC-only Tests

- HTTP overhead significant
- Differences in QUIC+HTTP vs QUIC-only differences varies
  → QUIC+HTTP performance not representative for QUIC-only results (and vice versa!)
- Traffic generator efficiency varies for same(!) implementations
  → Efficiency of traffic generators themselves impact results
- QUIC-only throughput limited
  - On sender side
  - By single-core CPU performance

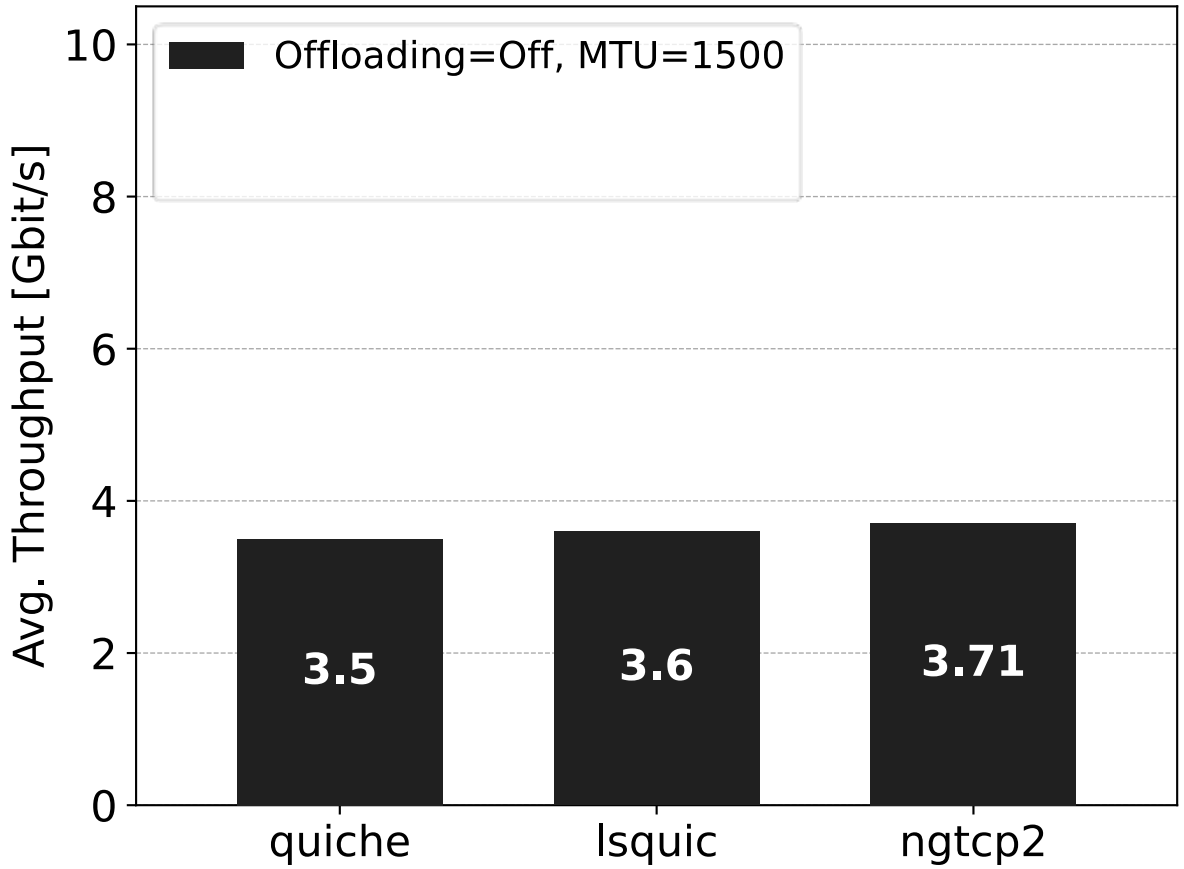
QUIC throughput performance across the network stack

Application level: QUIC+HTTP traffic ✓

Transport level: QUIC-only traffic ✓

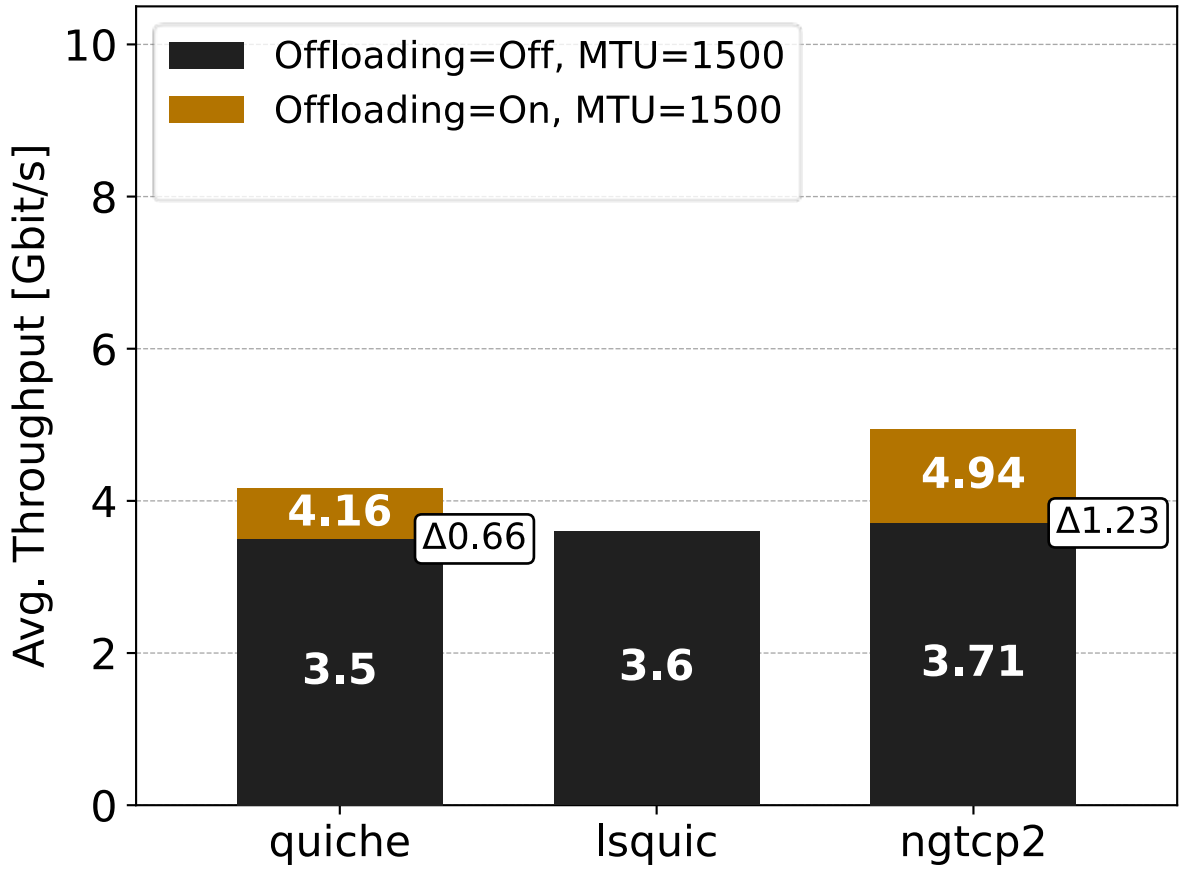
Lower level: Offloading & MTU

Offloading and Packet Size



QUIC -only

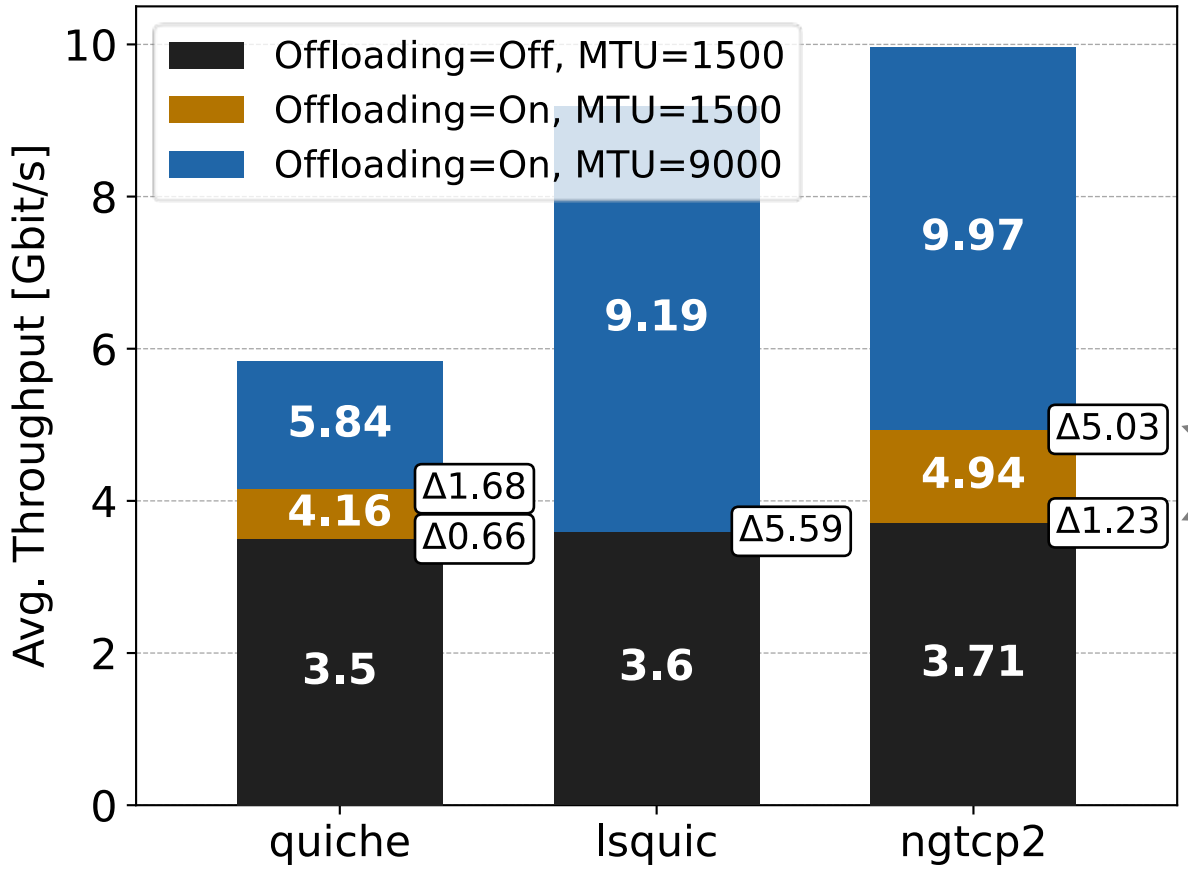
Offloading and Packet Size



QUIC -only

→ Generic offloading can increase throughput substantially

Offloading and Packet Size



QUIC -only

Offloading: 1.23 Gbit/s
Larger packets: 5.03 Gbit/s

⟶ “per QUIC packet”-processing overhead even greater

⟶ Efficient QUIC-specific offloading techniques required

Research Group Prof. Zitterbart (KIT)
Research Group Prof. Scheuermann (TUDa)

Summary & Conclusion

- Throughput performance varies drastically between
  - QUIC implementations and
  - Sender-receiver combinations
  - For both QUIC+HTTP and QUIC-only traffic

- Better distinction between QUIC and QUIC+HTTP performance results
  → Performance results not representative for each other

- Generic offloading improves performance substantial
- “Per-QUIC packet” processing overhead even greater
  → QUIC-specific offloading features required

- Support for a common QUIC traffic generator across implementations (similar to iperf3 for TCP/UDP)
  → Better comparability

- Possible solution: More efficient implementations + Reduce context switches by
  - Moving QUIC into the Kernel (one common & tuned QUIC socket)
  - Circumvent Kernel network stack (DPDK, XDP, …)