

Kernel Bypass Surgery: A Viable Procedure for Maximizing QUIC Bandwidth?

Johannes Späth¹, Stefan Lachnit¹, Marcel Kempf¹, Kilian Holzinger¹, Georg Carle¹, Johannes Zirngibl²

Friday 22nd May, 2026

IEEE/IFIP NOMS 2026
Rome, Italy



¹ Technical University of Munich

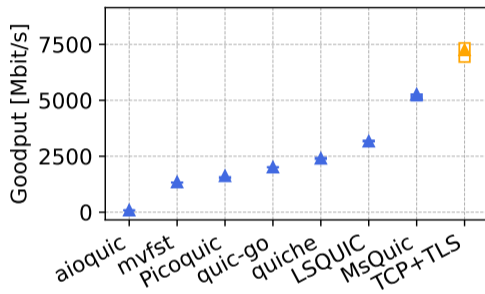


MAX PLANCK INSTITUTE
FOR INFORMATICS

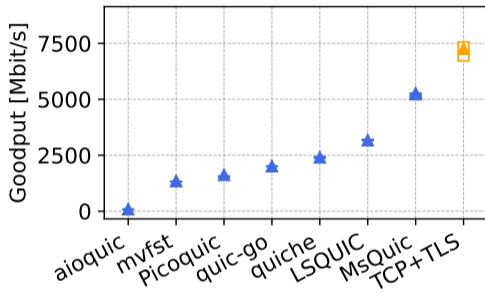
² Max Planck Institute for Informatics

- QUIC is increasingly used for **high-throughput** scenarios
 - File downloads
 - Media over QUIC (IETF)
 - SMB over QUIC (Microsoft)

- QUIC is increasingly used for **high-throughput** scenarios
 - File downloads
 - Media over QUIC (IETF)
 - SMB over QUIC (Microsoft)
- **But:** performance across QUIC libraries varies a lot and is often less than what TCP achieves

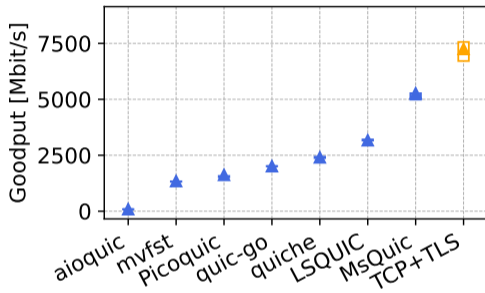


- QUIC is increasingly used for **high-throughput** scenarios
 - File downloads
 - Media over QUIC (IETF)
 - SMB over QUIC (Microsoft)
- **But:** performance across QUIC libraries varies a lot and is often less than what TCP achieves
- QUIC is built on top of UDP and (usually) implemented in user space



RQ 1 Can kernel bypass with DPDK improve QUIC performance?

- QUIC is increasingly used for **high-throughput** scenarios
 - File downloads
 - Media over QUIC (IETF)
 - SMB over QUIC (Microsoft)
- **But:** performance across QUIC libraries varies a lot and is often less than what TCP achieves
- QUIC is built on top of UDP and (usually) implemented in user space



RQ 1 Can kernel bypass with DPDK improve QUIC performance?

RQ 2 How does kernel bypass compare to segmentation offload (GSO/GRO)?

RQ 3 Does performance suffer when deploying QUIC on SR-IOV virtual functions?

RFC 9000 (2021)

Work	Year	High-Bandw.	Offl.	DPDK	#Libraries
[Megyesi 16]	2016	✗	✗	✗	1
[Yang 20]	2020	✓	○	✗	4
[Tyunyayev 22]	2022	✓	✗	✓	4
[König 23]	2023	✓	✗	✗	6
[Jaeger 23]	2023	✓	✓	✗	6
[Kempf 24]	2024	✓	✓	✗	7
[Zhang 24]	2024	○	✓	✗	1
[König 25]	2025	✓	✓	✗	5
This Work	2026	✓	✓	✓	4

[Megyesi 16] Péter Megyesi et al. "How quick is QUIC?". 2016.

[Yang 20] Xiangrui Yang et al. "Making QUIC Quicker With NIC Offload". 2020.

[Tyunyayev 22] Nikita Tyunyayev et al. "A High-Speed QUIC Implementation". 2022.

[König 23] Michael König et al. "QUIC(k) Enough in the Long Run? Sustained Throughput Performance of QUIC Implementations,". 2023.

[Jaeger 23] Benedikt Jaeger et al. "QUIC on the Highway: Evaluating Performance on High-rate Links". 2023.

[Kempf 24] Marcel Kempf et al. "QUIC on the Fast Lane: Extending Performance Evaluations on High-rate Links". 2024.

[Zhang 24] Xumiao Zhang et al. "QUIC is not Quick Enough over Fast Internet". 2024.

[König 25] Michael König et al. "Examining the Heterogeneous Throughput Performance Landscape of QUIC Implementations". 2025.

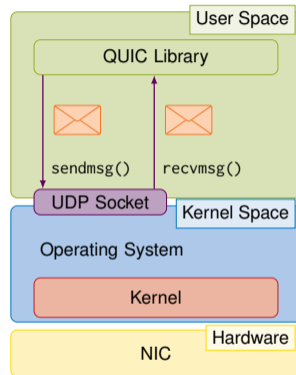
- Related work has shown a packet I/O bottleneck for common QUIC libraries [**Yang 20**, **Jaeger 23**, **Kempf 24**]

[**Yang 20**] Xiangrui Yang et al. "Making QUIC Quicker With NIC Offload". 2020.

[**Jaeger 23**] Benedikt Jaeger et al. "QUIC on the Highway: Evaluating Performance on High-rate Links". 2023.

[**Kempf 24**] Marcel Kempf et al. "QUIC on the Fast Lane: Extending Performance Evaluations on High-rate Links". 2024.

- Related work has shown a packet I/O bottleneck for common QUIC libraries [Yang 20, Jaeger 23, Kempf 24]
- QUIC is (usually) implemented in user space
 - ACK handling in user space
 - Context switches

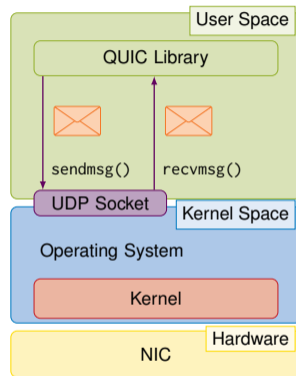


[Yang 20] Xiangrui Yang et al. "Making QUIC Quicker With NIC Offload". 2020.

[Jaeger 23] Benedikt Jaeger et al. "QUIC on the Highway: Evaluating Performance on High-rate Links". 2023.

[Kempf 24] Marcel Kempf et al. "QUIC on the Fast Lane: Extending Performance Evaluations on High-rate Links". 2024.

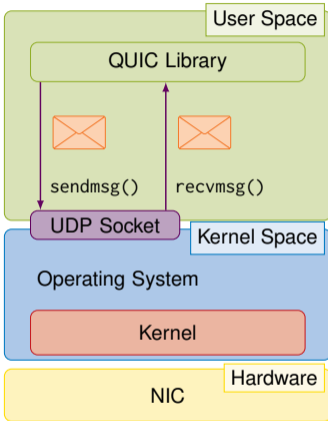
- Related work has shown a packet I/O bottleneck for common QUIC libraries [Yang 20, Jaeger 23, Kempf 24]
- QUIC is (usually) implemented in user space
 - ACK handling in user space
 - Context switches
- QUIC runs on top of UDP
 - Support for offloading techniques is rather new and has to be added by the implementation



[Yang 20] Xiangrui Yang et al. "Making QUIC Quicker With NIC Offload". 2020.

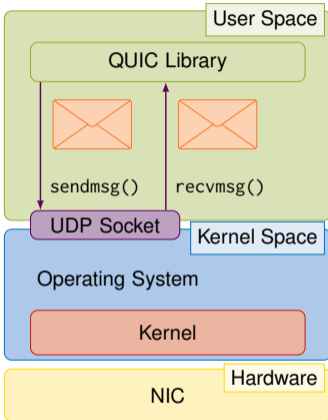
[Jaeger 23] Benedikt Jaeger et al. "QUIC on the Highway: Evaluating Performance on High-rate Links". 2023.

[Kempf 24] Marcel Kempf et al. "QUIC on the Fast Lane: Extending Performance Evaluations on High-rate Links". 2024.



Problems

- One system call per message send/receive
- Message size limited by MTU

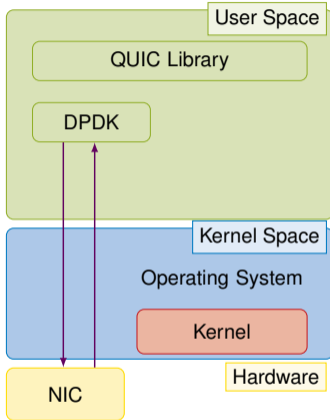


Problems

- One system call per message send/receive
- Message size limited by MTU

Segmentation Offload (UDP GSO/GRO)

- Allow large messages (> MTU, up to 64 KiB) to traverse the kernel network stack
- Segmentation/reassembly at the NIC (ideally with HW support)
- **Prerequisites:** NIC/driver + appl. support



Problems





- One system call per message send/receive
- Message size limited by MTU





Segmentation Offload (UDP GSO/GRO)





- Allow large messages (> MTU, up to 64 KiB) to traverse the kernel network stack
- Segmentation/reassembly at the NIC (ideally with HW support)
- **Prerequisites:** NIC/driver + appl. support

Kernel Bypass with DPDK





- Communicate with the NIC directly from user space (using DPDK)
- **Prerequisites:** full NIC access, polling-based appl. architecture

Implementation	Repository	Language	Integration Details
MsQuic	 microsoft/msquic	C	Datapath abstraction → Custom DPDK datapath implementation
LSQUIC	 litespeedtech/lsquic	C	
quiche	 cloudflare/quiche	Rust	
Picoquic	 privateoctopus/picoquic	C	

Implementation	Repository	Language	Integration Details
MsQuic	 microsoft/msquic	C	Datapath abstraction → Custom DPDK datapath implementation
LSQUIC	 litespeedtech/lsquic	C	Integration via single-threaded run-to-completion model
quiche	 cloudflare/quiche	Rust	Integration via quiche's C wrapper → No library changes necessary
Picoquic	 privateoctopus/picoquic	C	

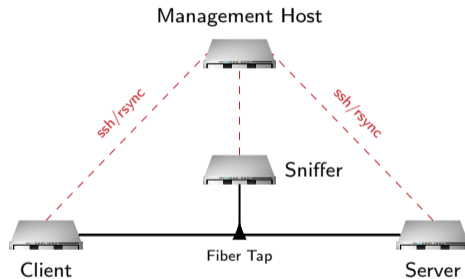
Implementation	Repository	Language	Integration Details
MsQuic	 microsoft/msquic	C	Datapath abstraction → Custom DPDK datapath implementation
LSQUIC	 litespeedtech/lsquic	C	Integration via single-threaded run-to-completion model
quiche	 cloudflare/quiche	Rust	Integration via quiche's C wrapper → No library changes necessary
Picoquic	 privateoctopus/picoquic	C	Existing picoquic-DPDK implementation [Tyunyayev 22] → Replaced existing packet loop with an active polling loop

[Tyunyayev 22] Nikita Tyunyayev et al. "A High-Speed QUIC Implementation". 2022.

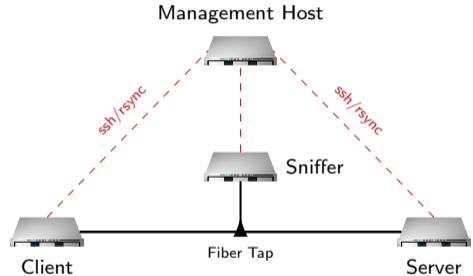
Implementation	Repository	Language	Integration Details
MsQuic	 microsoft/msquic	C	Datapath abstraction → Custom DPDK datapath implementation
LSQUIC	 litespeedtech/lsquic	C	Integration via single-threaded run-to-completion model
quiche	 cloudflare/quiche	Rust	Integration via quiche's C wrapper → No library changes necessary
Picoquic	 privateoctopus/picoquic	C	Existing picoquic-DPDK implementation [Tyunyayev 22] → Replaced existing packet loop with an active polling loop

→ **Key Insight:** Custom integration of DPDK per library required

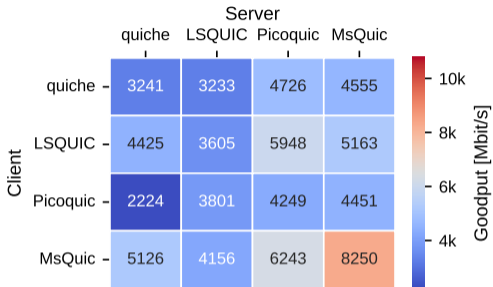
- Hardware testbed
- File transfers over direct 100 Gbit/s link, repeated runs



- Hardware testbed
- File transfers over direct 100 Gbit/s link, repeated runs
- Measurement framework based on Jaeger et al. **[Jaeger 23]**
- Extensions
 - DPDK support (metadata export, interface setup, ...)
 - Optical splitter support



Results: Implementation Matrices



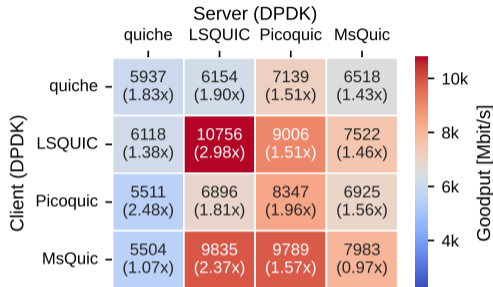
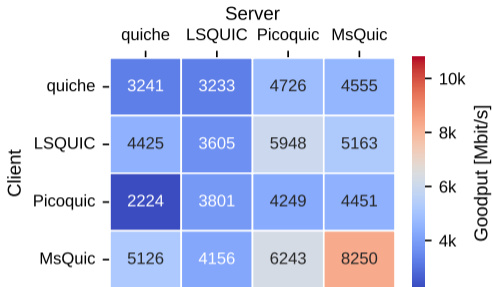
Measurements

- 8 GiB file downloads
- 50 repetitions

Insights

- Significant differences across combinations
- MsQuic is the fastest client

Results: Implementation Matrices



Measurements

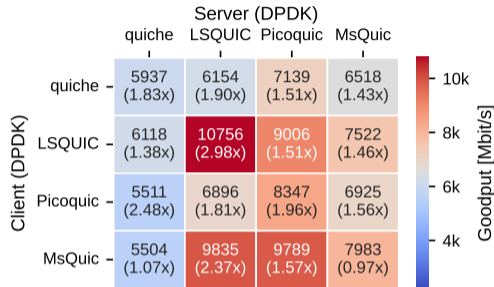
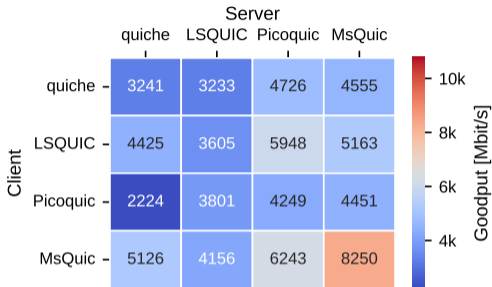
- 8 GiB file downloads
- 50 repetitions

Insights

- Significant differences across combinations
- MsQuic is the fastest client

- LSQUIC-DPDK achieves highest goodput and speedup
- MsQuic is slightly slower with DPDK

Results: Implementation Matrices



Measurements

- 8 GiB file downloads
- 50 repetitions

Insights

- Significant differences across combinations
- MsQuic is the fastest client

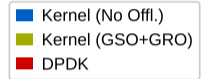
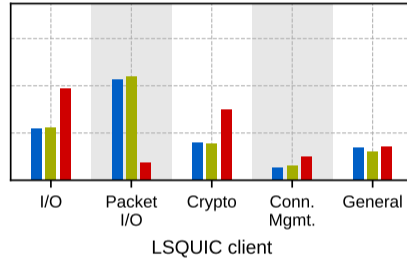
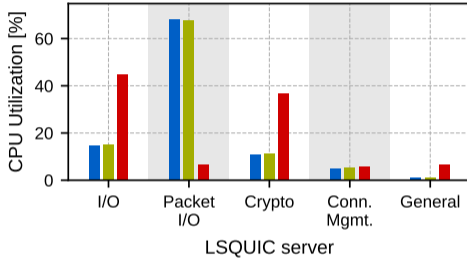
- LSQUIC-DPDK achieves highest goodput and speedup
- MsQuic is slightly slower with DPDK

→ **Key Insight: High speedup potential by using DPDK**

Results: In-Depth Analysis – LSQUIC



LSQUIC-DPDK Goodput: 10.8 Gbit/s (speedup: 2.98×)



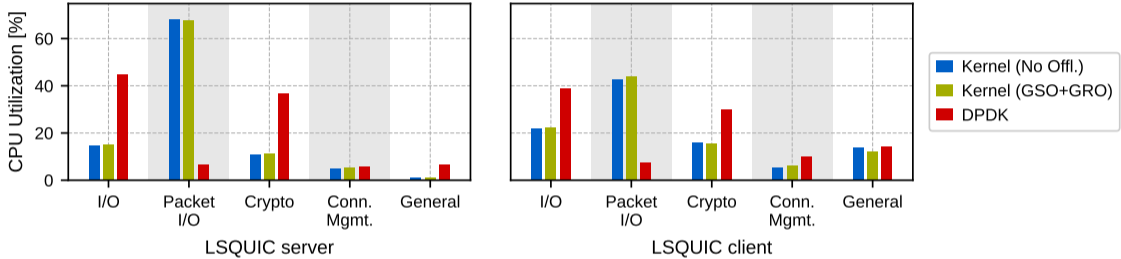
Measurements

- 1 GiB file downloads
- 10 repetitions

Results: In-Depth Analysis – LSQUIC



LSQUIC-DPDK Goodput: 10.8 Gbit/s (speedup: 2.98×)



Measurements

- 1 GiB file downloads
- 10 repetitions

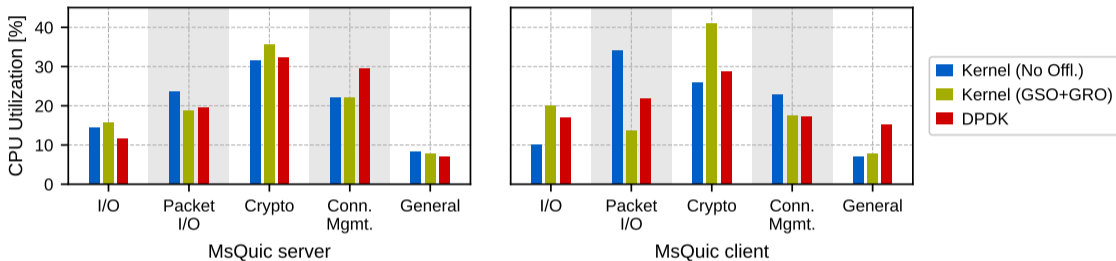
Insights

- Offloading has no effect (not supported by LSQUIC)
- DPDK decreases packet I/O by up to 90%

→ **Key Insight:** Kernel bypass can solve the packet I/O bottleneck

Results: In-Depth Analysis – MsQuic

MsQuic-DPDK Goodput: 8.0 Gbit/s (speedup: 0.97 \times)



Measurements

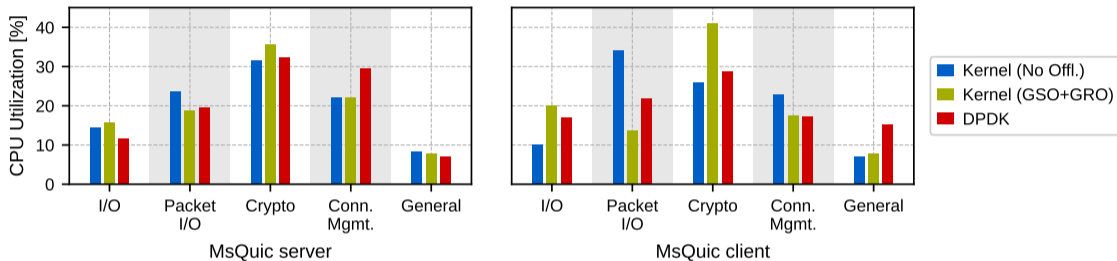
- 1 GiB file downloads
- 10 repetitions

Insights

- GSO/GRO and DPDK both decrease packet I/O
- Offloading has more effect (especially at the client)

Results: In-Depth Analysis – MsQuic

MsQuic-DPDK Goodput: 8.0 Gbit/s (speedup: 0.97 \times)



Measurements

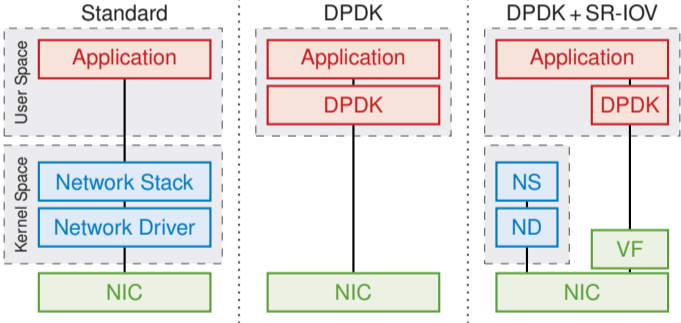
- 1 GiB file downloads
- 10 repetitions

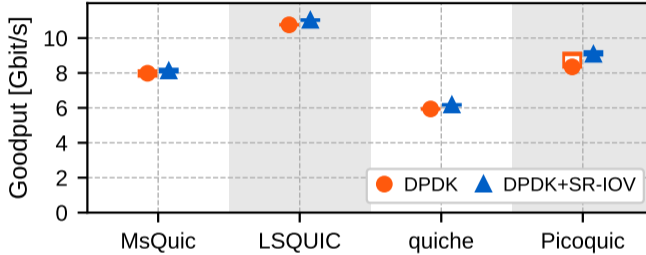
Insights

- GSO/GRO and DPDK both decrease packet I/O
- Offloading has more effect (especially at the client)

→ **Key Insight:** Segmentation Offload can also solve the packet I/O bottleneck

Motivation: Deployability of QUIC-DPDK implementations, no full NIC access required





→ **Key Insight:** No goodput degradation of DPDK with SR-IOV

Conclusions



Key Insights

- DPDK can solve QUIC's packet I/O bottleneck
- **But:** requires custom integration per library, achievable speedup varies
- GSO/GRO can achieve similar goodput
- DPDK can effectively be deployed with SR-IOV

Conclusions



Key Insights

- DDPK can solve QUIC's packet I/O bottleneck
- **But:** requires custom integration per library, achievable speedup varies
- GSO/GRO can achieve similar goodput
- DDPK can effectively be deployed with SR-IOV

Artifacts

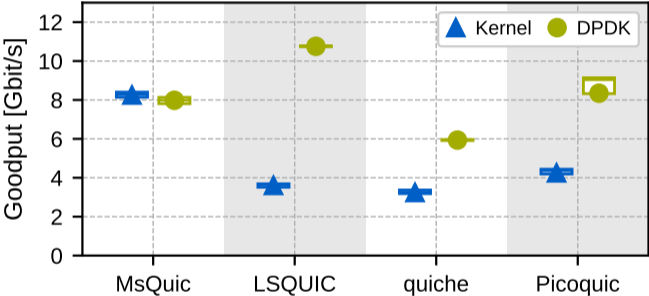
- Measurement framework, results
 - 🔗 [tumi8/quic-bypass-paper](#)
- QUIC-DDPK implementations
 - 🔗 [tumi8/msquic-dpdk](#)
 - 🔗 [tumi8/lrsquic-dpdk](#)
 - 🔗 [tumi8/quiche-dpdk](#)
 - 🔗 [tumi8/picoquic-dpdk](#)



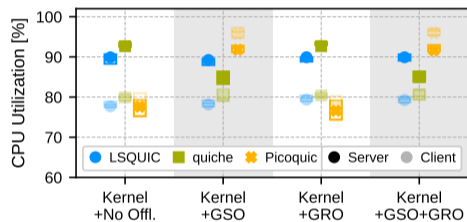
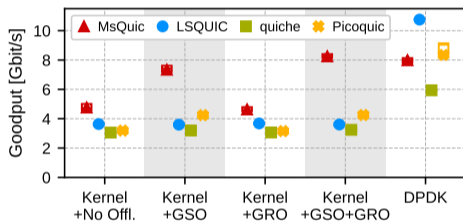
Paper PDF

Questions? – spaethj@net.in.tum.de

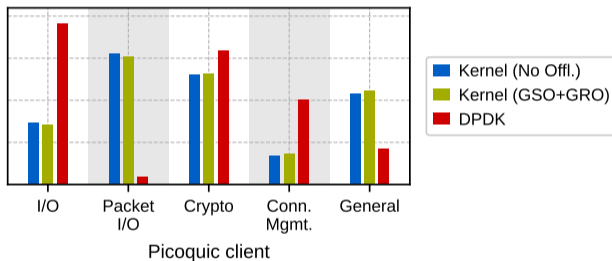
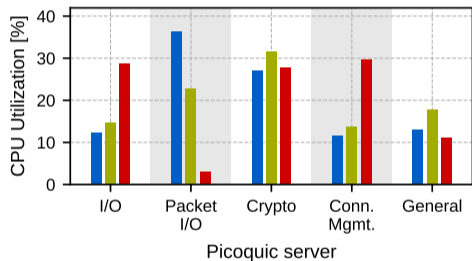
Backup Slides



Offloading Speedup



Picoquic-DPDK Goodput: 8.3 Gbit/s (speedup: 1.96 \times)



quiche-DPDK Goodput: 5.9 Gbit/s (speedup: 1.83 \times)

