Parallelizing Packet Processing in Container Overlay Networks

(EuroSys '21)

Jiaxin Lei[1], Manish Munikar[2], Kun Suo[3], Hui Lu[1], Jia Rao[2]

[1] *SUNY Binghamton*
[2] *The University of Texas at Arlington*
[3] *Kennesaw State University*

Containers are everywhere

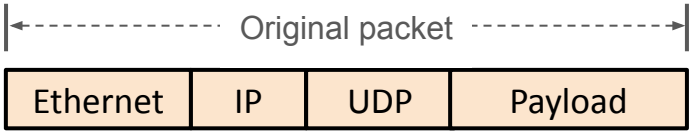
- Containers are revolutionizing cloud.
  - Lightweight OS-level virtualization

Containers are everywhere

- Containers are revolutionizing cloud.
  - Lightweight OS-level virtualization
- Containers communicate using overlay network
  - VXLAN encapsulation

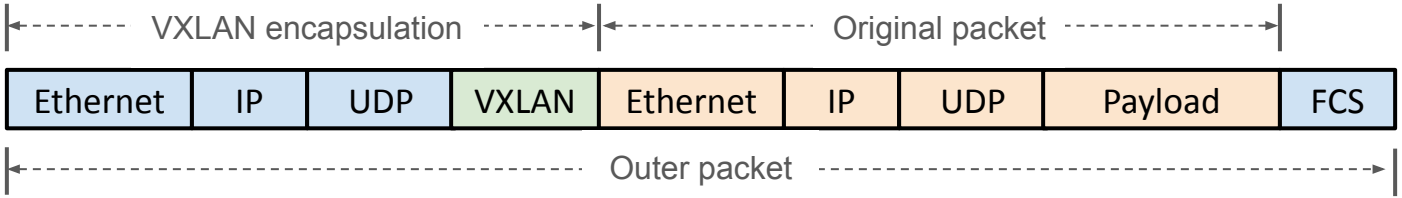
Containers are everywhere

- Containers are revolutionizing cloud.
  - Lightweight OS-level virtualization
- Containers communicate using overlay network
  - VXLAN encapsulation



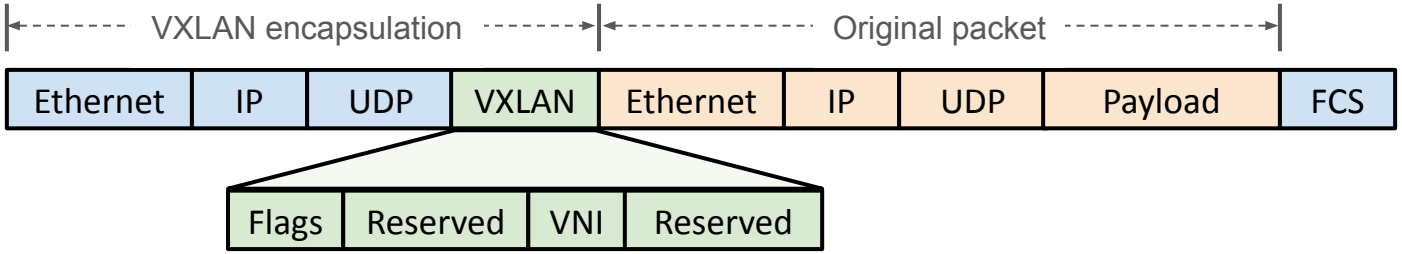
Containers are everywhere

- Containers are revolutionizing cloud.
  - Lightweight OS-level virtualization
- Containers communicate using overlay network
  - VXLAN encapsulation



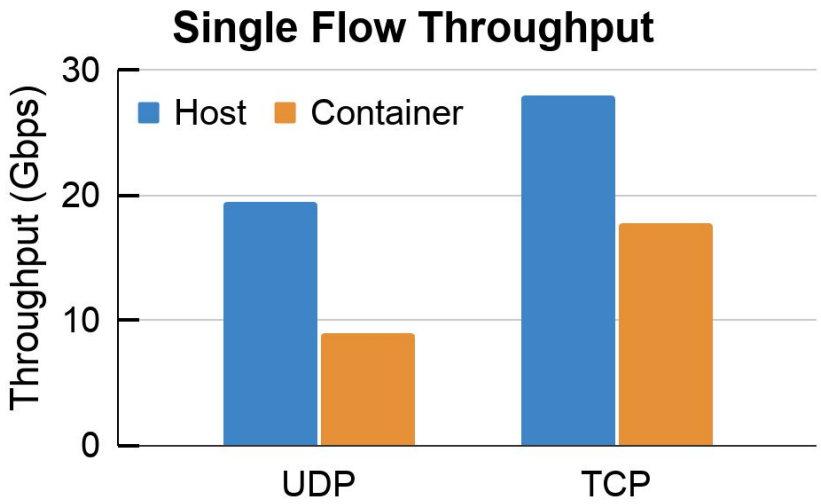
Containers are everywhere

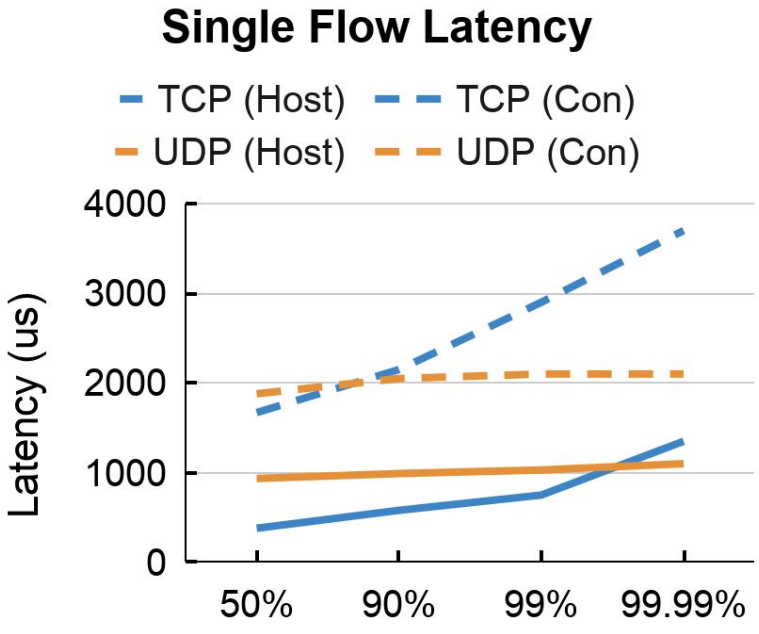
- Containers are revolutionizing cloud.
  - Lightweight OS-level virtualization
- Containers communicate using overlay network
  - VXLAN encapsulation



Overlay network is slow

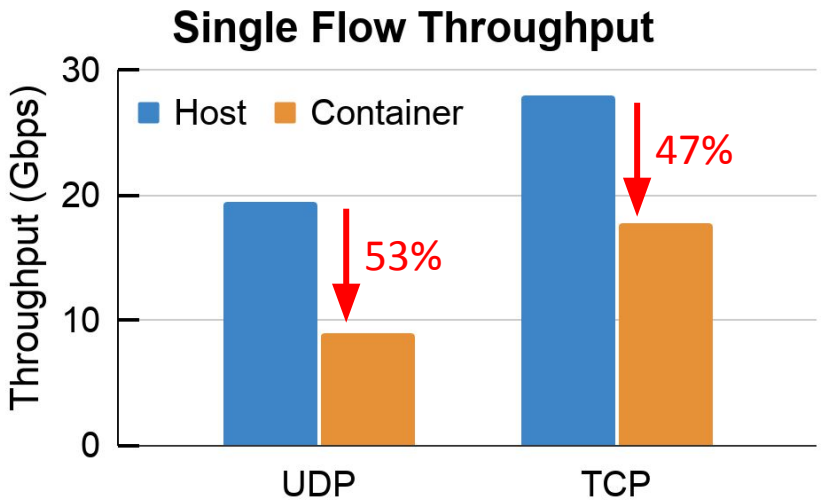
- Compared to host, overlay network has:

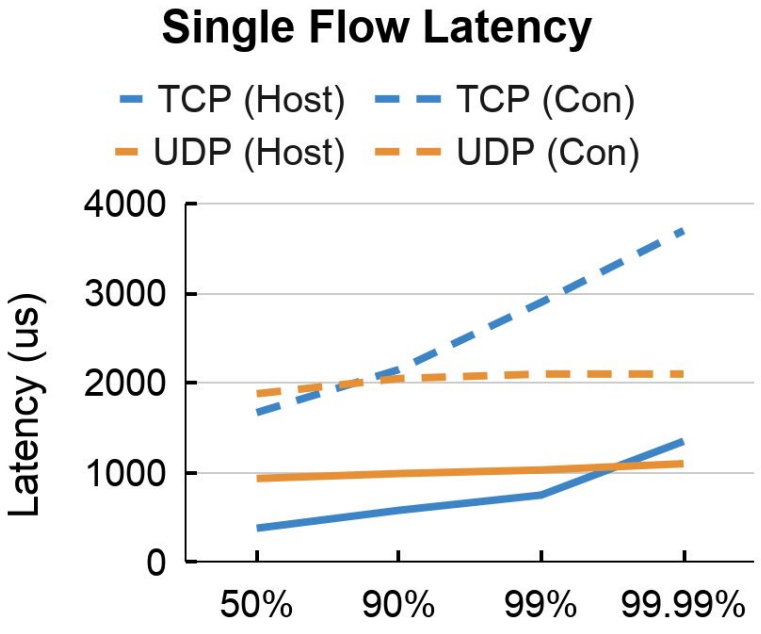




Overlay network is slow

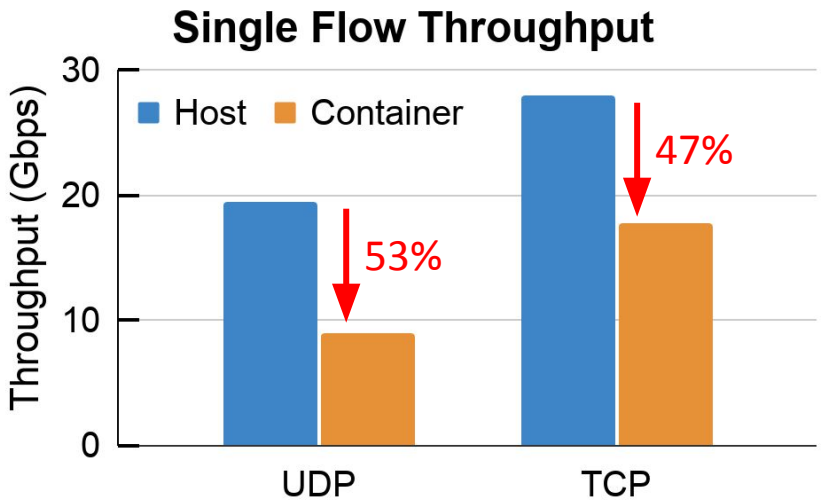
- Compared to host, overlay network has:
  - Half the throughput

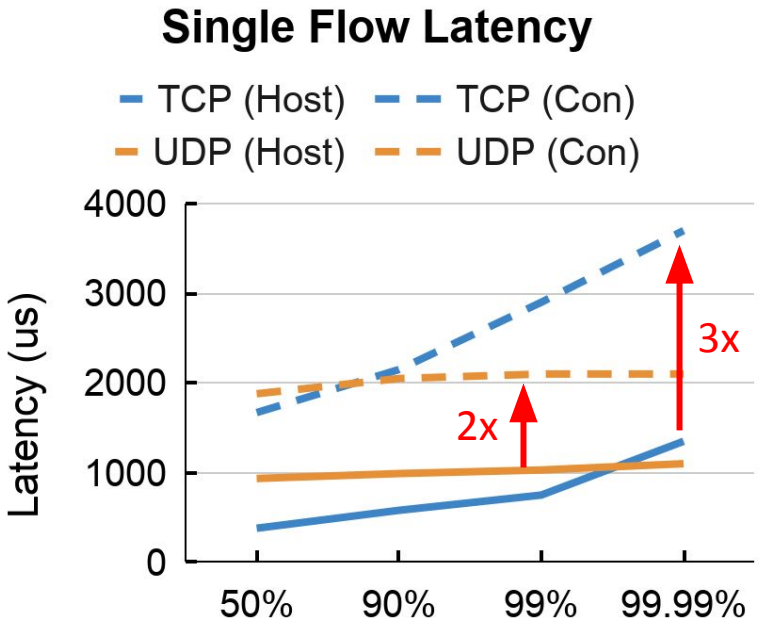




Overlay network is slow

- Compared to host, overlay network has:
  - Half the throughput
  - Double per-packet latency

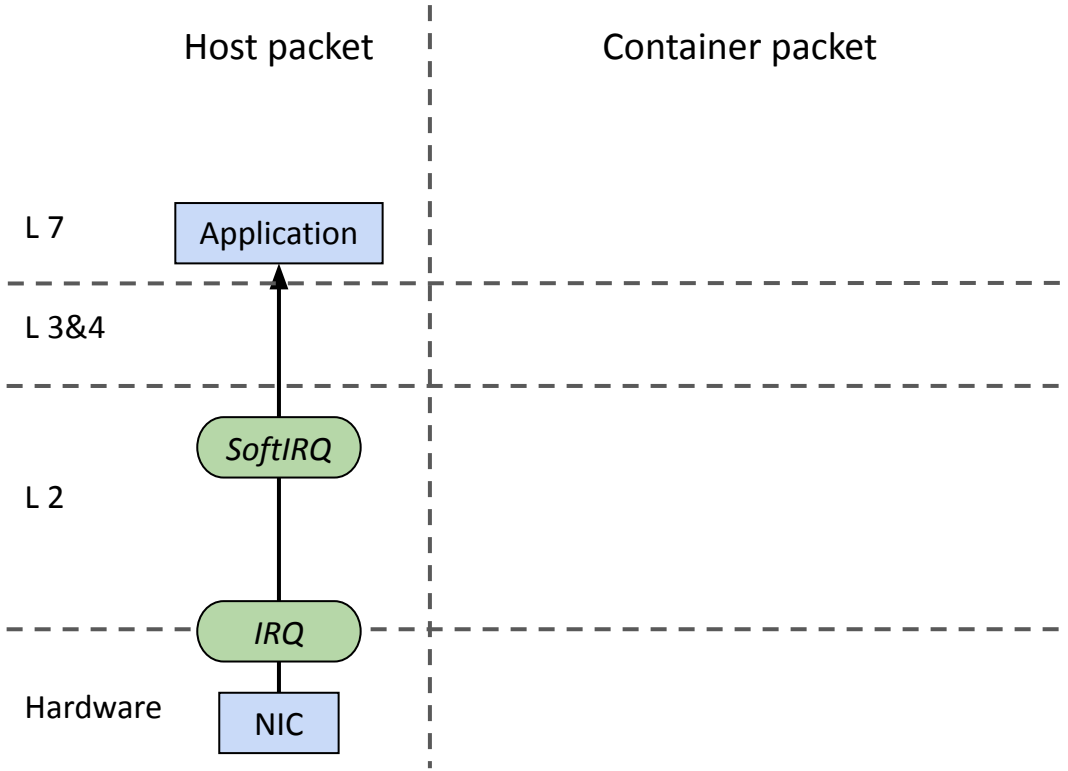




Why are overlay networks so slow?

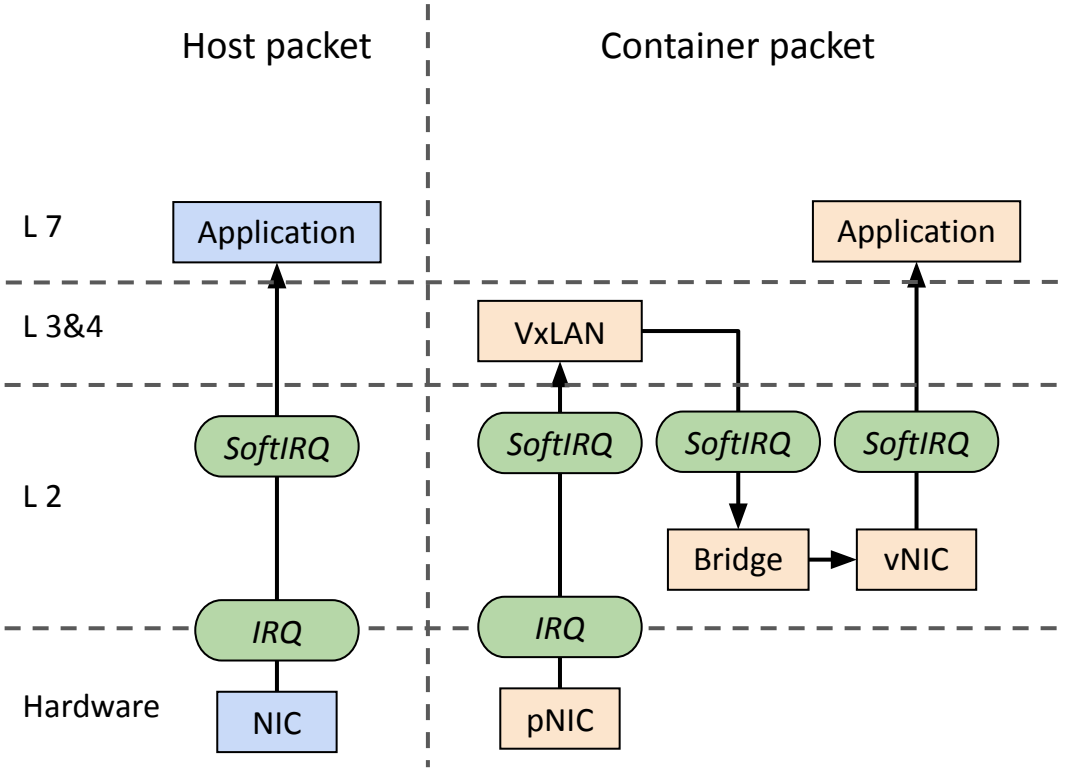
Why are overlay networks so slow?

- Host packet
  - 1 IRQ + 1 SoftIRQ



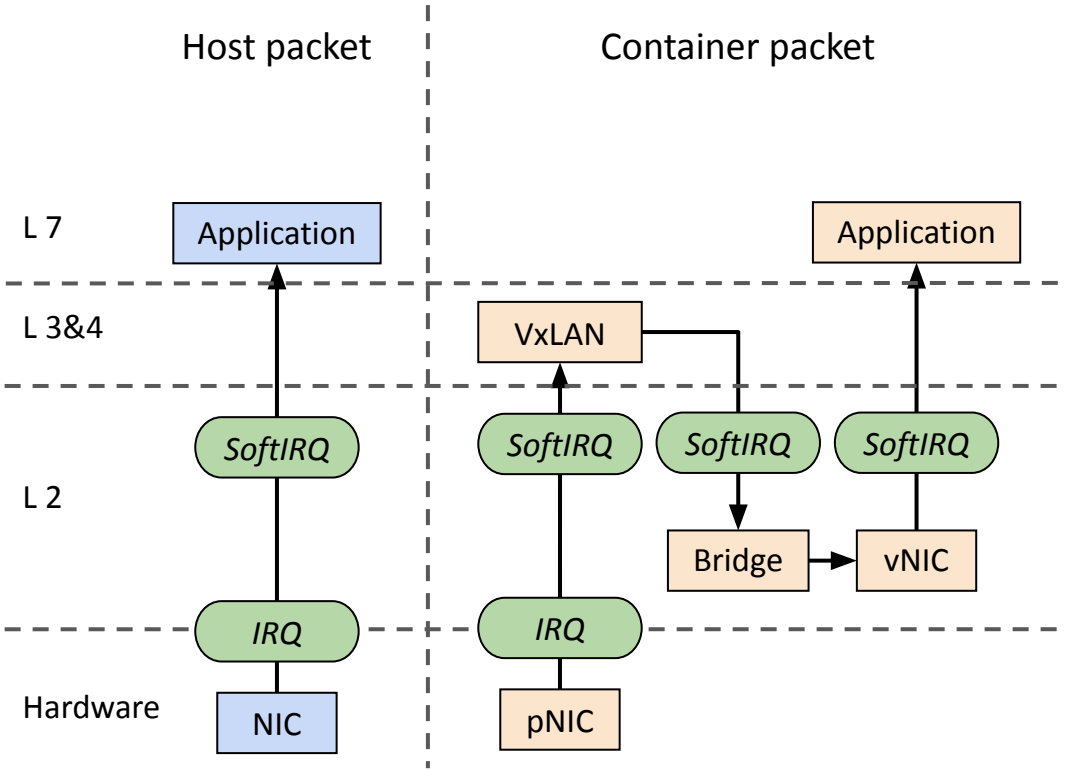
Why are overlay networks so slow?

- Host packet
  - 1 IRQ + 1 SoftIRQ
- Container packet
  - 1 IRQ + 3 SoftIRQs



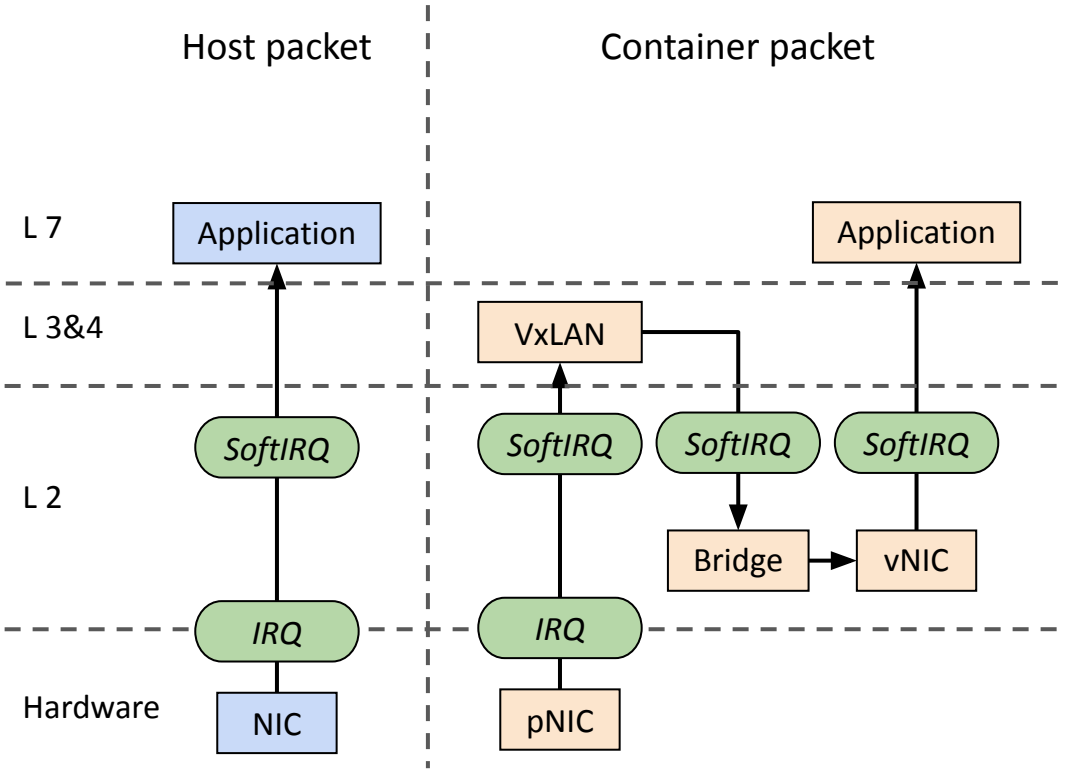
Why are overlay networks so slow?

1. Prolonged datapath
   - Multiple virtual devices to traverse for each packet
   - 3x more softirq



Why are overlay networks so slow?

1. Prolonged datapath
   - Multiple virtual devices to traverse for each packet
   - 3x more softirq
2. Serialized softirq execution
   - Load imbalance
   - Longer queue delay



Existing optimizations

Existing optimizations

- Kernel-bypass [DPDK, mTCP, TAS]
  - ✅ Avoid OS overheads; custom minimal network stack
  - ❌ Loose security, compatibility

Existing optimizations

- Kernel-bypass [DPDK, mTCP, TAS]
  - ✅ Avoid OS overheads; custom minimal network stack
  - ❌ Loose security, compatibility
- Connection-level metadata manipulation [Slim, FreeFlow]
  - ✅ Avoids overhead of virtual devices; as fast as host
  - ❌ Limited scope and scalability; cannot support dataplane policies

Existing optimizations

- Kernel-bypass [DPDK, mTCP, TAS]
  - ✅ Avoid OS overheads; custom minimal network stack
  - ❌ Loose security, compatibility
- Connection-level metadata manipulation [Slim, FreeFlow]
  - ✅ Avoids overhead of virtual devices; as fast as host
  - ❌ Limited scope and scalability; cannot support dataplane policies
- Hardware offload [Mellanox ASAP^2, AccelNet, RDMA]
  - ✅ Fastest; completely avoids CPU overheads
  - ❌ Requires hardware upgrade; limited flexibility

Our approach

FALCON = **F**ast **a**nd Ba**l**anced **Co**ntainer **N**etworking

Key idea: Leverage multicore architecture to accelerate overlay packet processing

Our approach

FALCON = **F**ast **a**nd Ba**l**anced **Co**ntainer **N**etworking

Key idea: Leverage multicore architecture to accelerate overlay packet processing

- ✅ Software-based solution
- ✅ Full network isolation / flexibility
- ✅ Completely backward compatible
- ✅ Better performance

Design 1: Softirq Pipelining

Key idea: Pipeline different softirqs onto different cores

Design 1: Softirq Pipelining

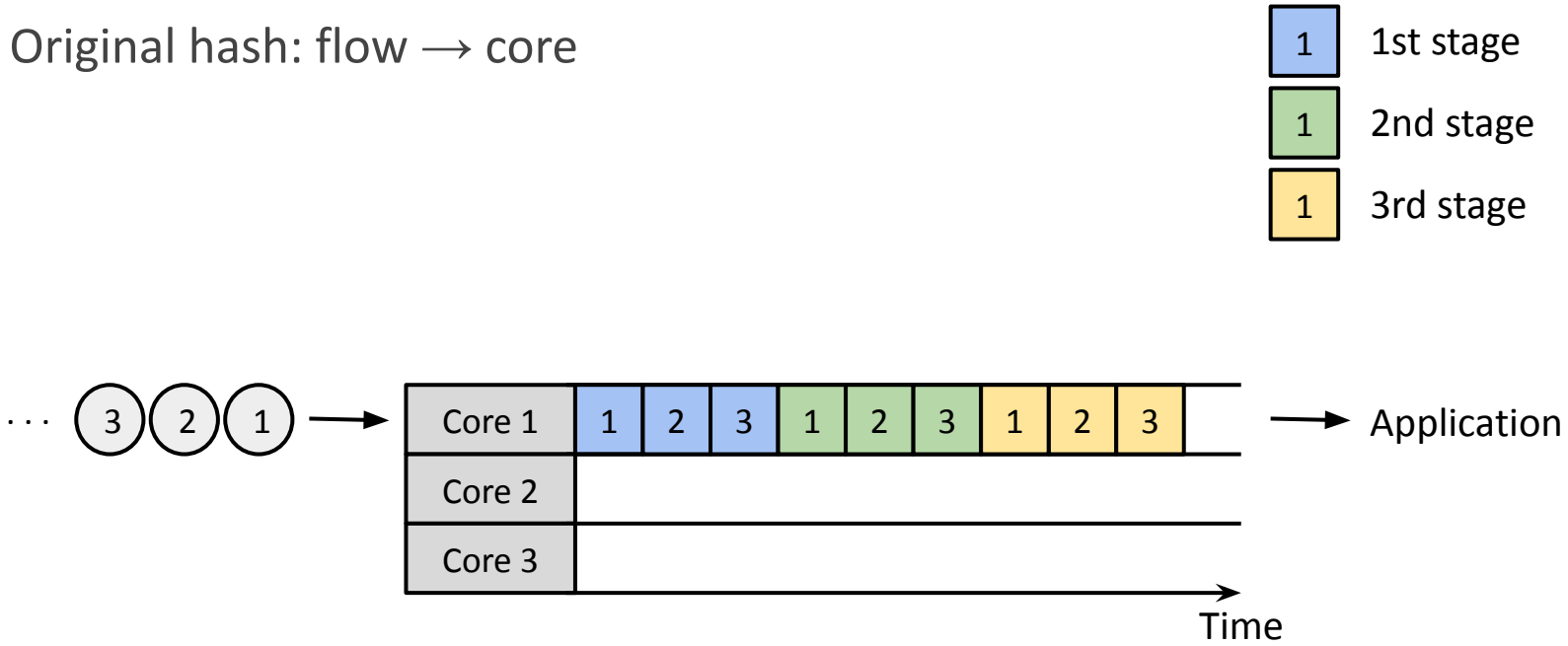
Key idea: Pipeline different softirqs onto different cores

- Original hash: flow → core

Design 1: Softirq Pipelining

Key idea: Pipeline different softirqs onto different cores

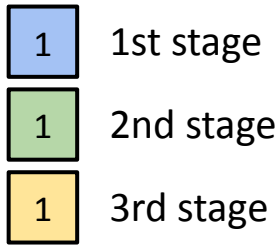
- Original hash: flow ⟶ core

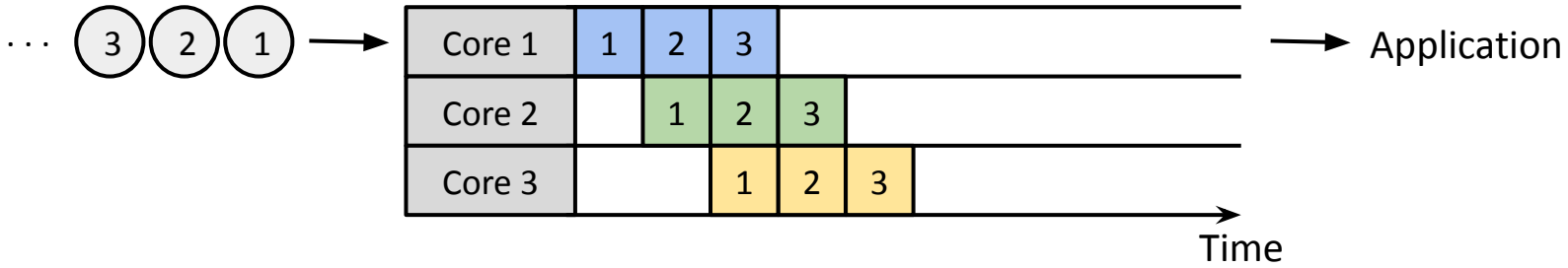


Design 1: Softirq Pipelining

Key idea: Pipeline different softirqs onto different cores

- Original hash: flow ⟶ core
- New hash: (flow, device) ⟶ core

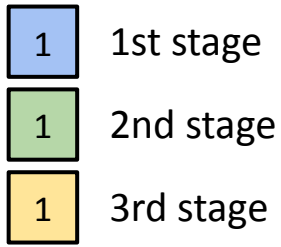


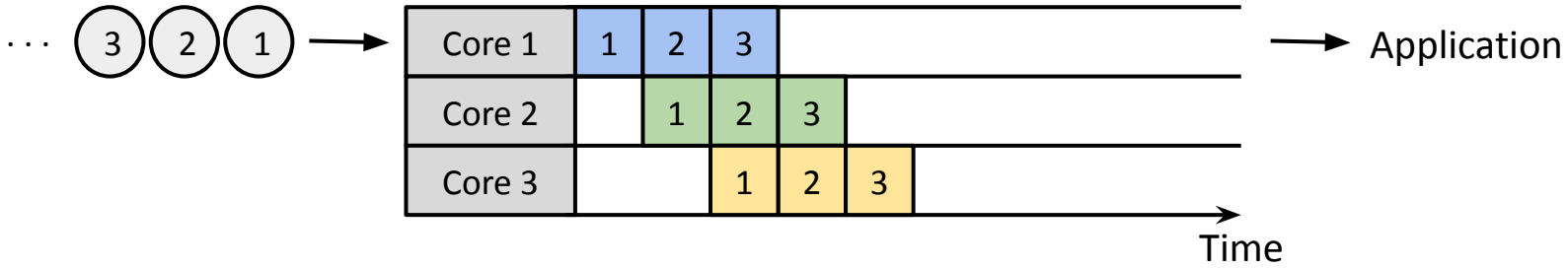


Design 1: Softirq Pipelining

Key idea: Pipeline different softirqs onto different cores

- Original hash: flow ⟶ core
- New hash: (flow, device) ⟶ core
- Order of packets is still preserved





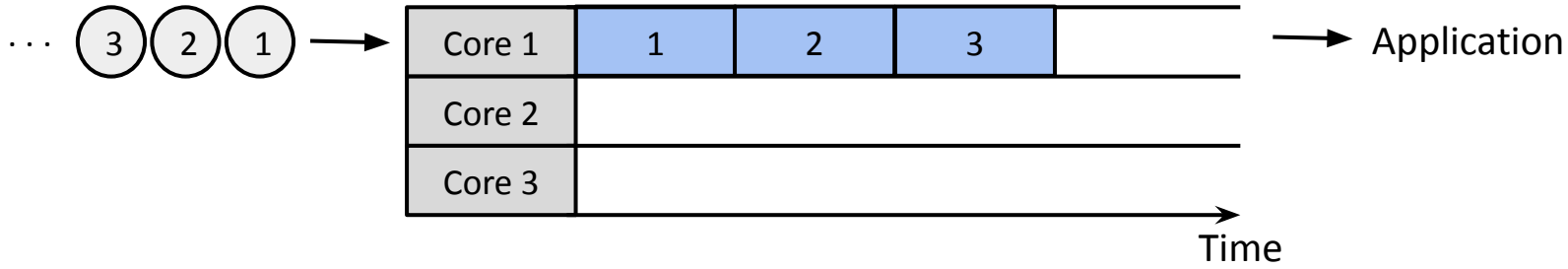
Design 2: Softirq Splitting

Key idea: Split one big softirq into two that can be pipelined

Design 2: Softirq Splitting

Key idea: Split one big softirq into two that can be pipelined

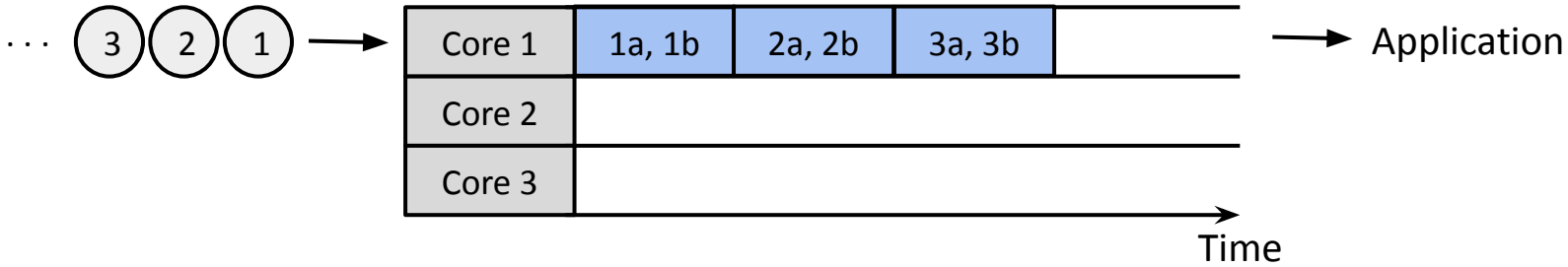
- Overlay TCP processing is heavily dominated by first stage



Design 2: Softirq Splitting

Key idea: Split one big softirq into two that can be pipelined

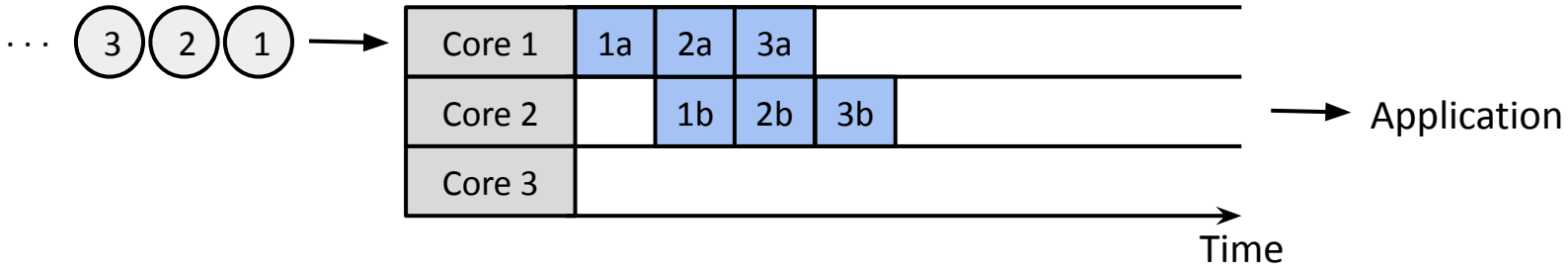
- Overlay TCP processing is heavily dominated by first stage
  - Two main functions: (a) SKB allocation, (b) GRO processing



Design 2: Softirq Splitting

Key idea: Split one big softirq into two that can be pipelined

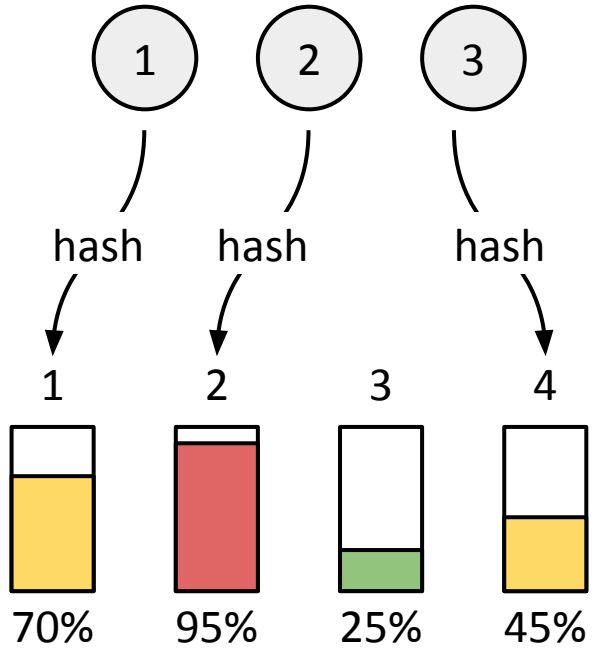
- Overlay TCP processing is heavily dominated by first stage
  - Two main functions: (a) SKB allocation, (b) GRO processing
- Split them by adding a softirq in the middle



Design 3: Softirq Balancing

Key idea: Try to dispatch softirqs on idle cores, else disable Falcon

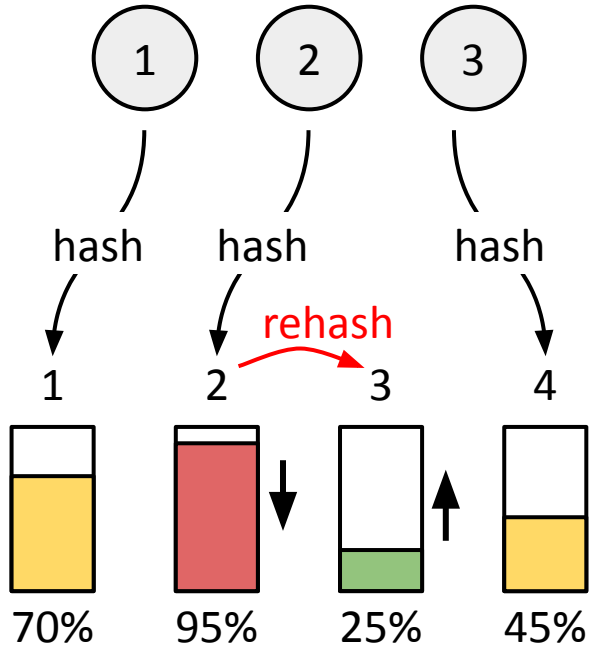
- Static hashing
  - Prone to load imbalance
  - Hurts performance if load is already high



Design 3: Softirq Balancing

Key idea: Try to dispatch softirqs on idle cores, else disable Falcon

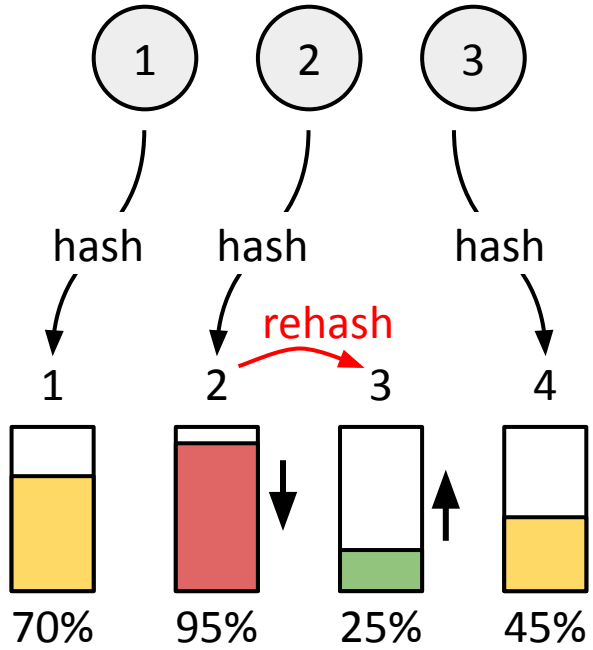
- Static hashing
  - Prone to load imbalance
  - Hurts performance if load is already high
- Dynamic rehashing
  - More balanced CPU utilization



Design 3: Softirq Balancing

Key idea: Try to dispatch softirqs on idle cores, else disable Falcon

- Static hashing
  - Prone to load imbalance
  - Hurts performance if load is already high
- Dynamic rehashing
  - More balanced CPU utilization
- Disable FALCON when overall system usage is high.



Evaluation — Setup

Hardware: Intel Xeon, 40 logical cores @ 2.2GHz, 128 GB RAM

NIC: Mellanox ConnectX-5 EN (100 Gbps)

Software: Ubuntu 18.04, with Linux kernel 5.4

Evaluation — Setup

Hardware: Intel Xeon, 40 logical cores @ 2.2GHz, 128 GB RAM

NIC: Mellanox ConnectX-5 EN (100 Gbps)

Software: Ubuntu 18.04, with Linux kernel 5.4

Comparison: FALCON vs. Container vs. Host

Evaluation — Setup

Hardware: Intel Xeon, 40 logical cores @ 2.2GHz, 128 GB RAM

NIC: Mellanox ConnectX-5 EN (100 Gbps)

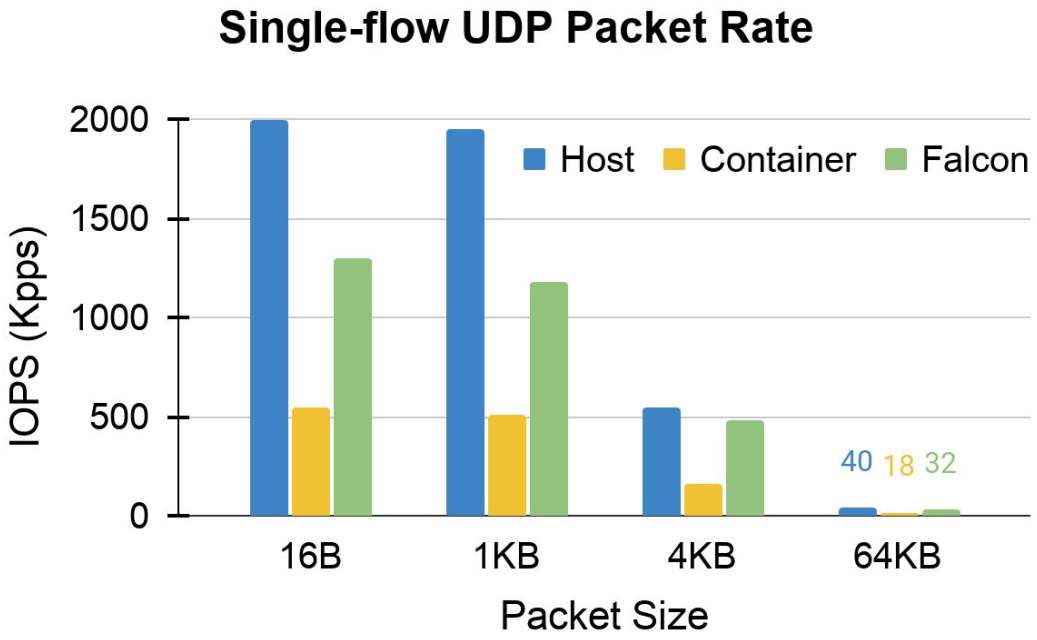
Software: Ubuntu 18.04, with Linux kernel 5.4

Comparison: FALCON vs. Container vs. Host

Experiments:

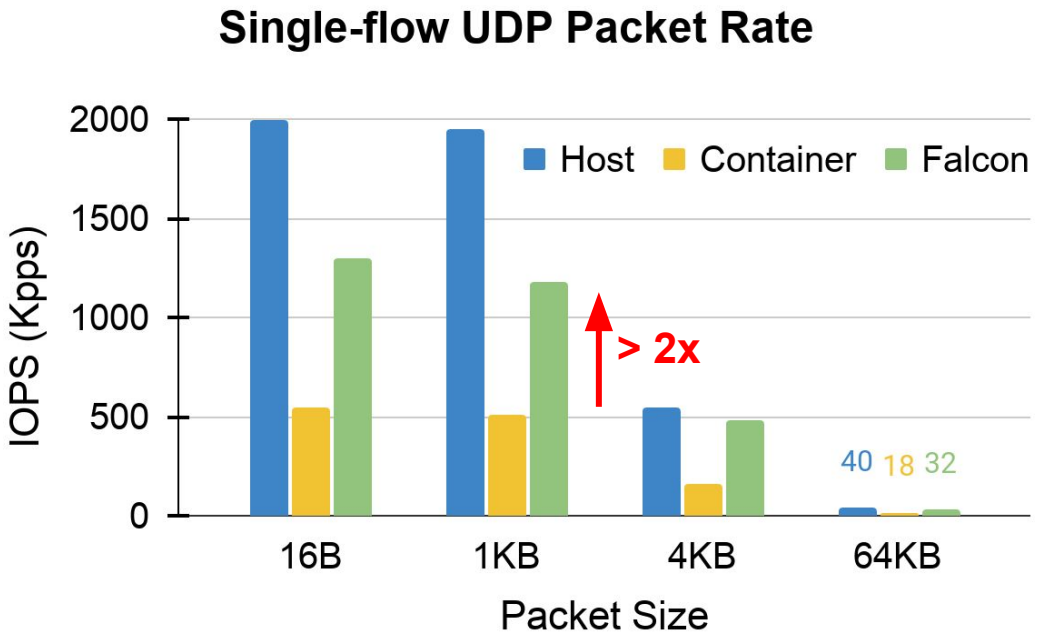
- Single-flow and multi-flow microbenchmarks
- Application benchmarks (CloudSuite web & data-caching)
- *[many others in the paper]*

Single-flow throughput



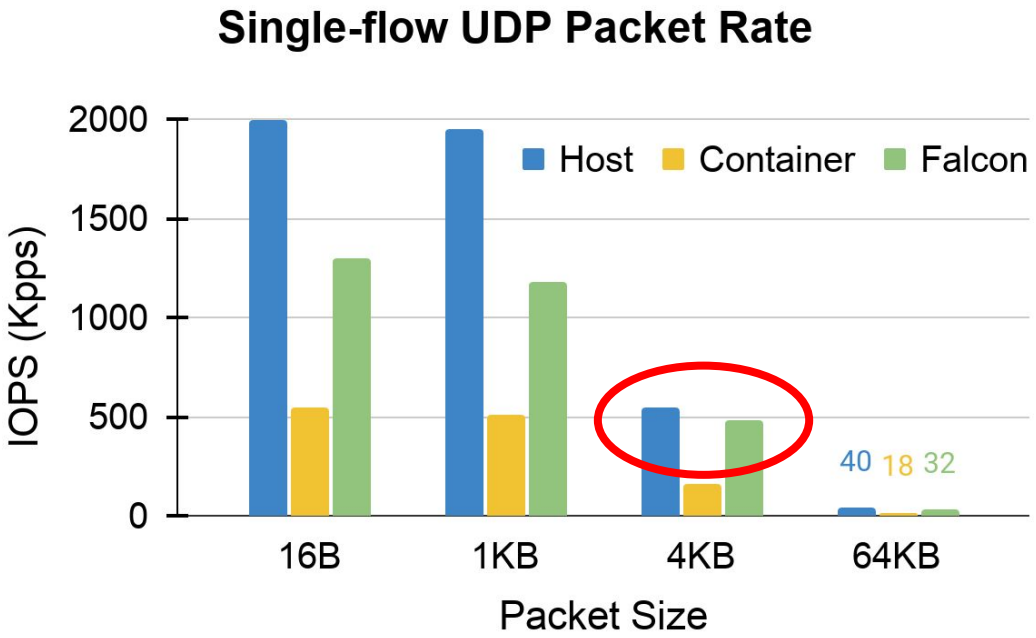
Single-flow throughput

- FALCON is results at more than 2x better packet rate than Container

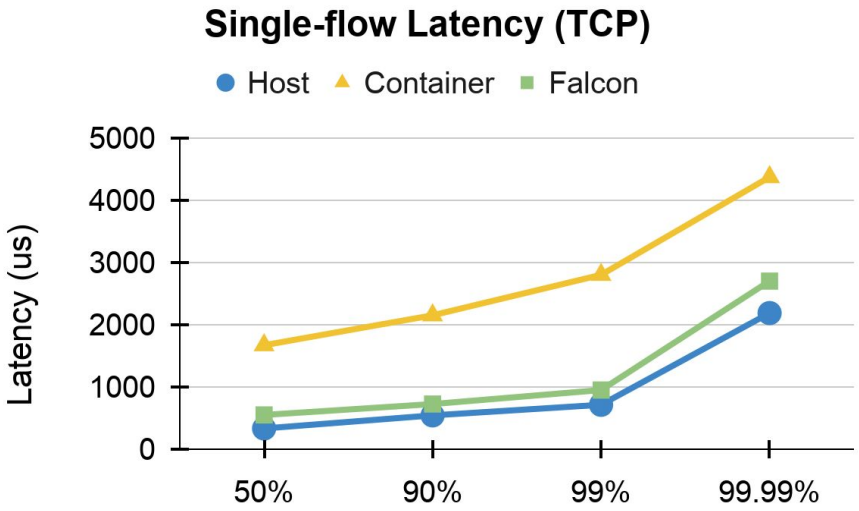


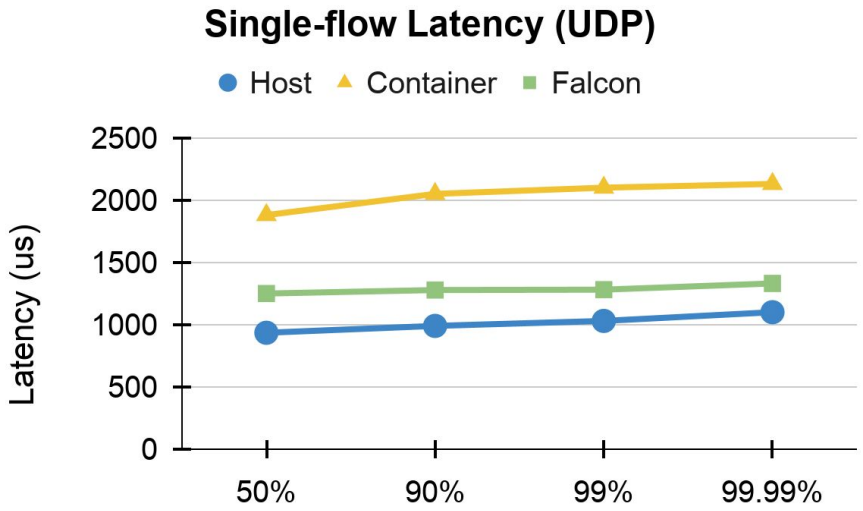
Single-flow throughput

- FALCON is results at more than 2x better packet rate than Container
- Closer to Host performance for large packet sizes



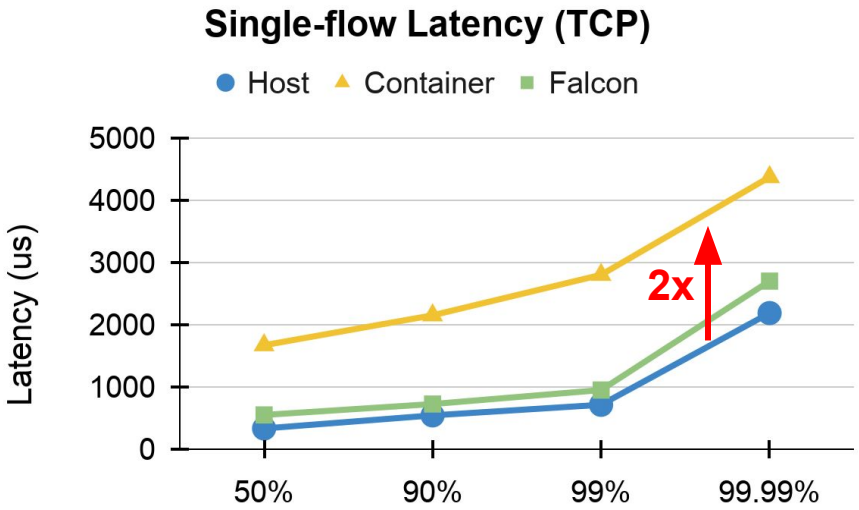
Single-flow latency

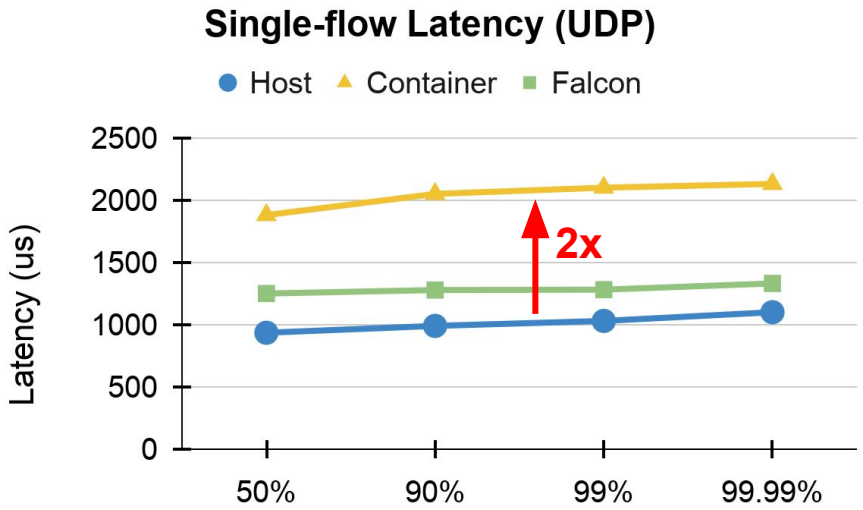




Single-flow latency

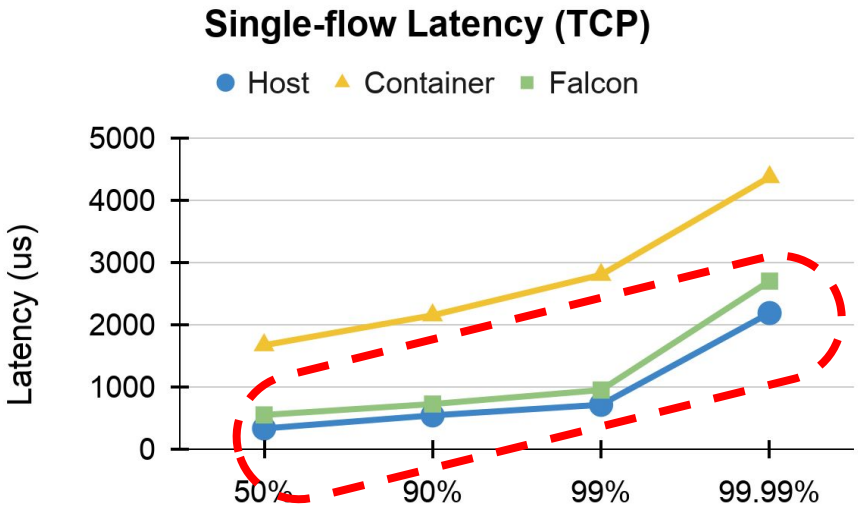
- Container latency is 2x of host

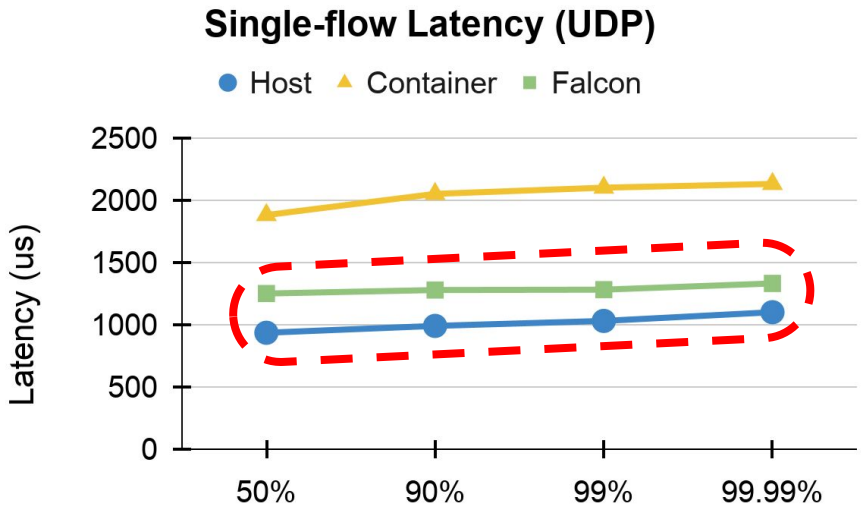




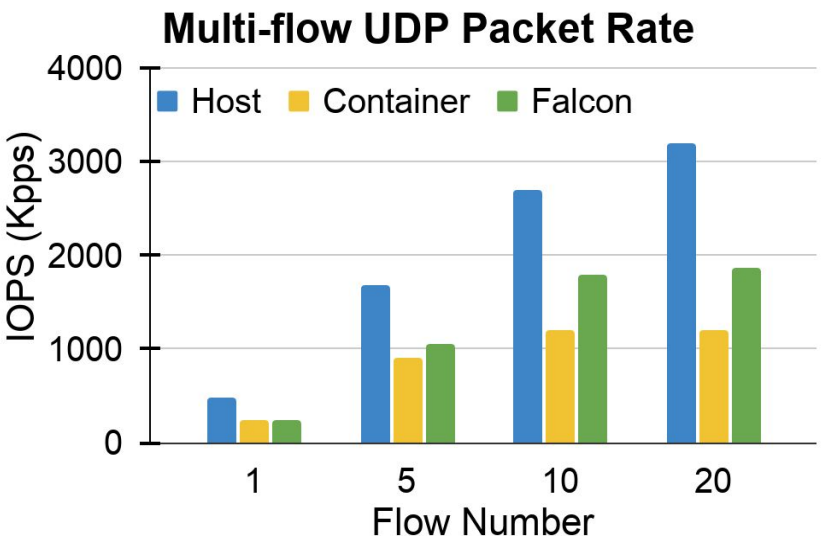
Single-flow latency

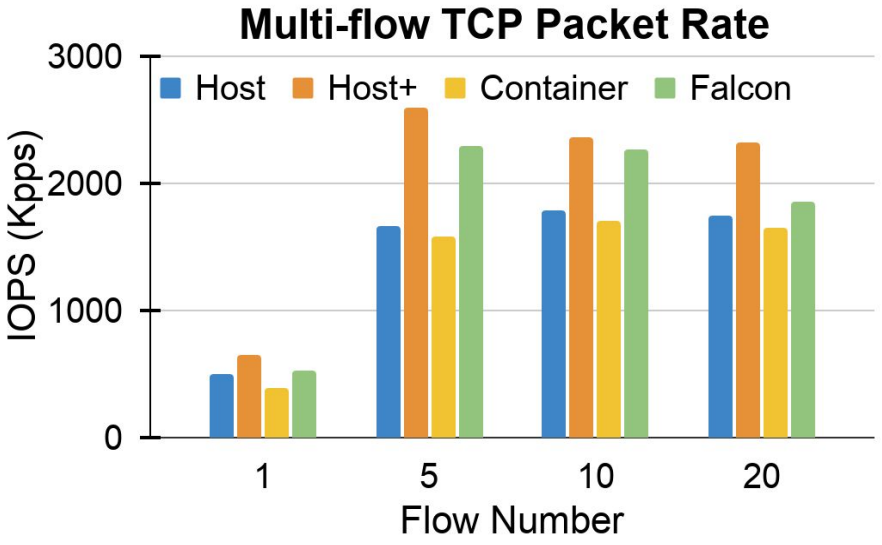
- Container latency is 2x of host
- FALCON achieves latency closer to host





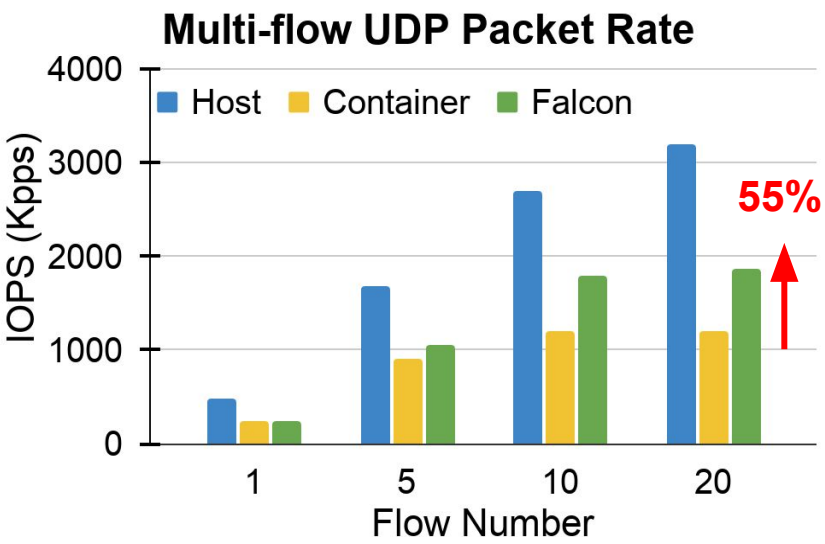
Multi-flow throughput

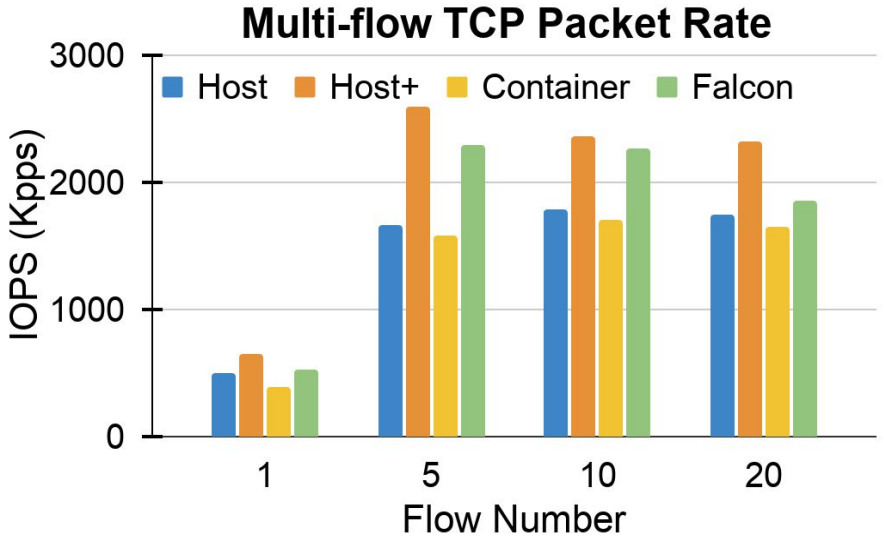




Multi-flow throughput

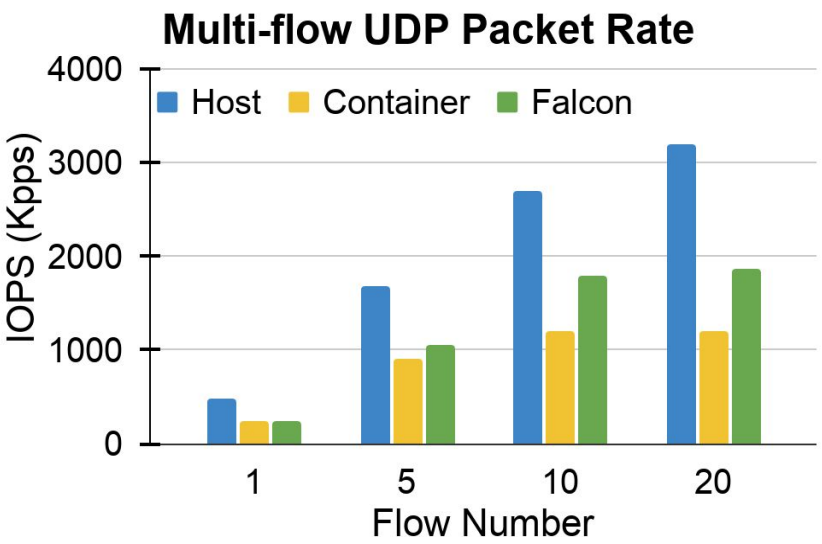
- UDP: Improves overlay network as much as 55%

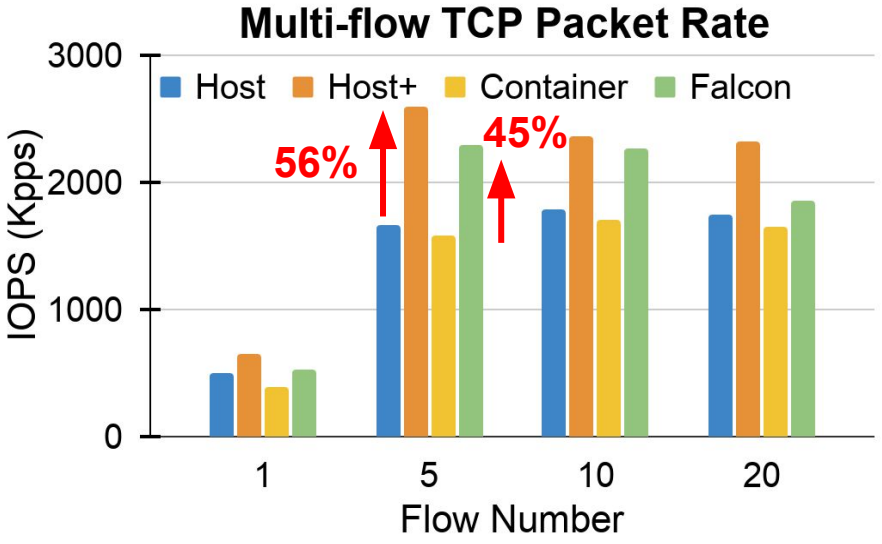




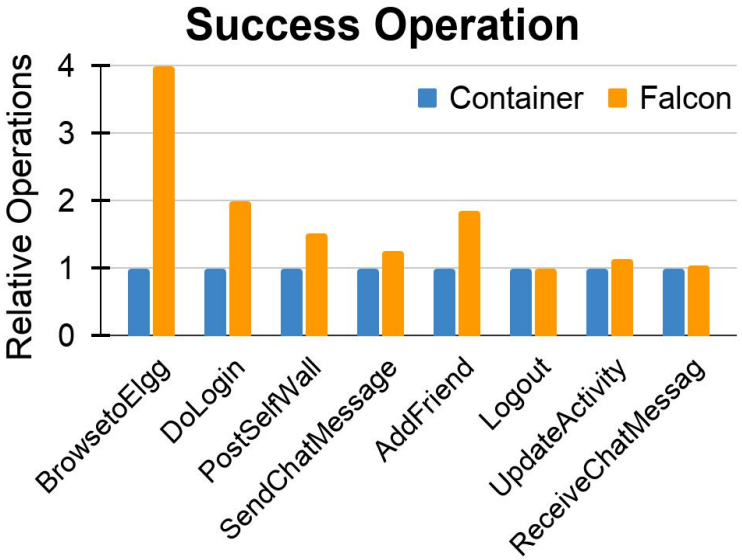
Multi-flow throughput

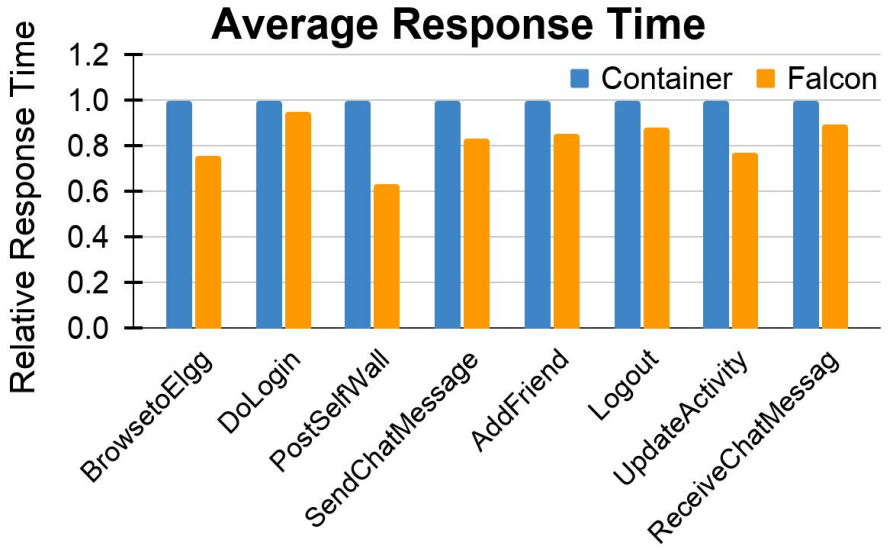
- UDP: Improves overlay network as much as 55%
- TCP: Improves overlay network by 45% (host network by 56%)





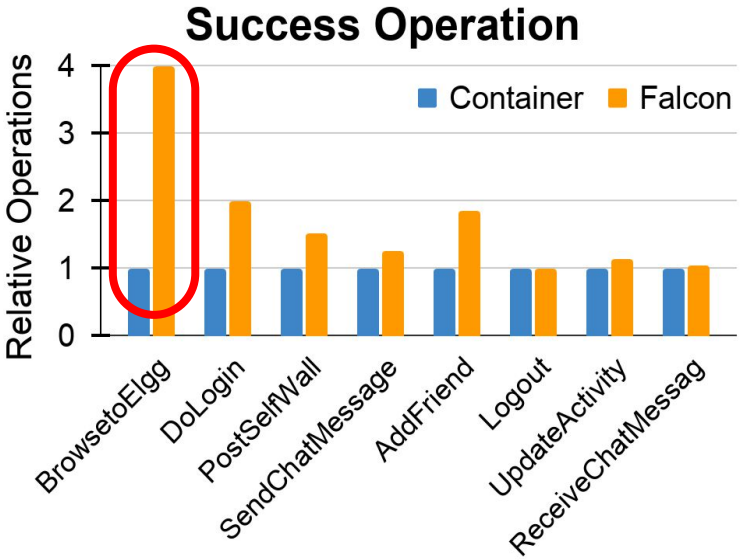
Cloud benchmarks: Web serving

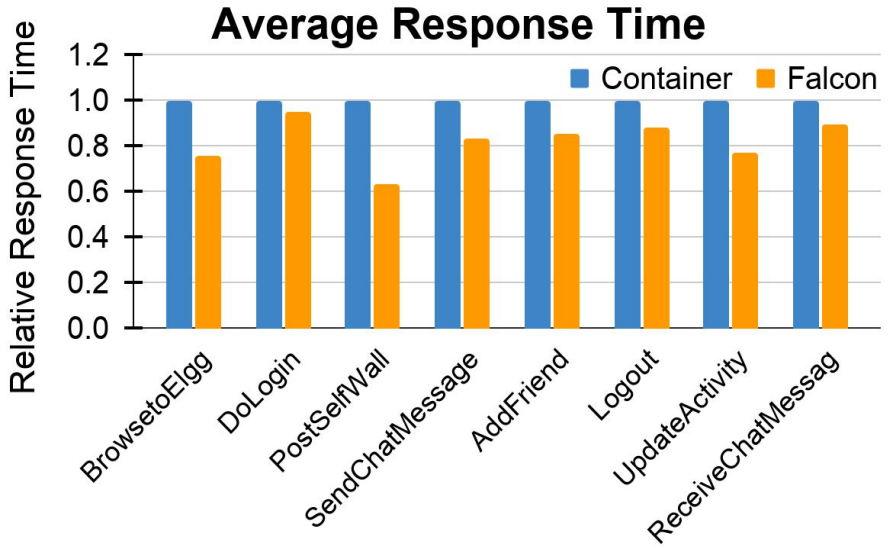




Cloud benchmarks: Web serving

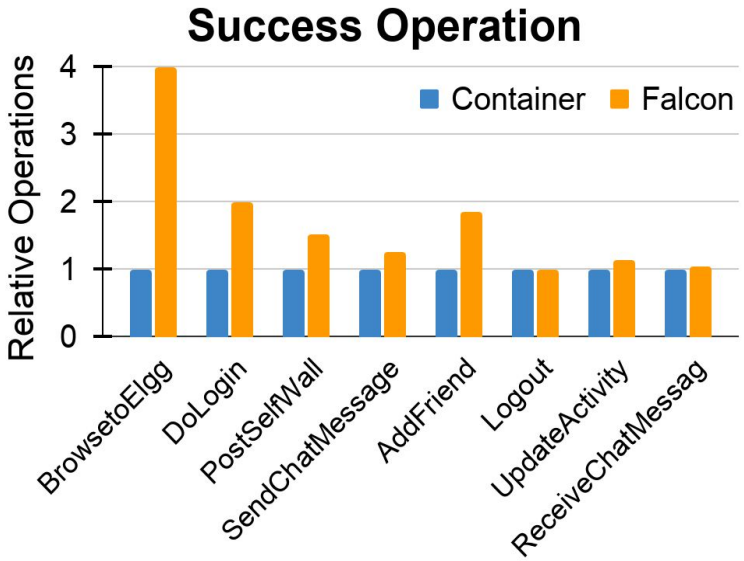
- Throughput improved by up to 300%

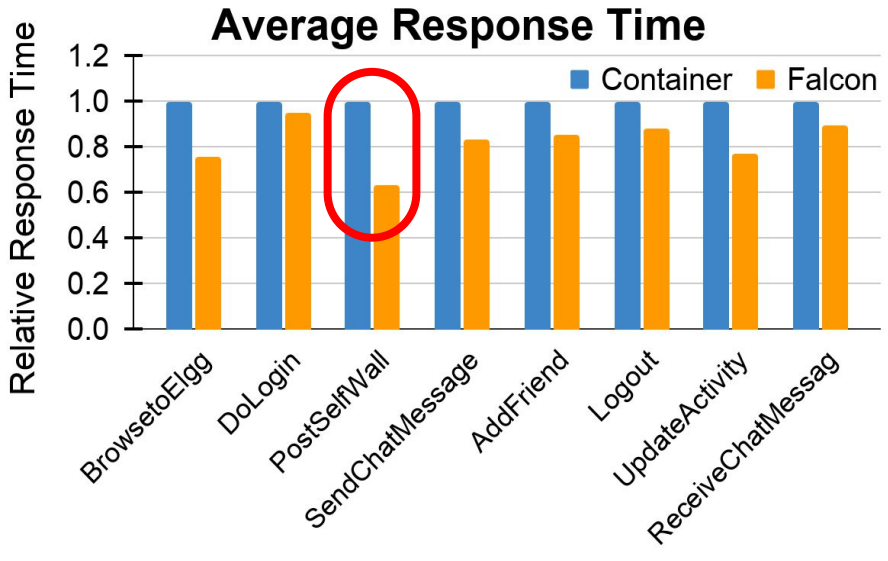




Cloud benchmarks: Web serving

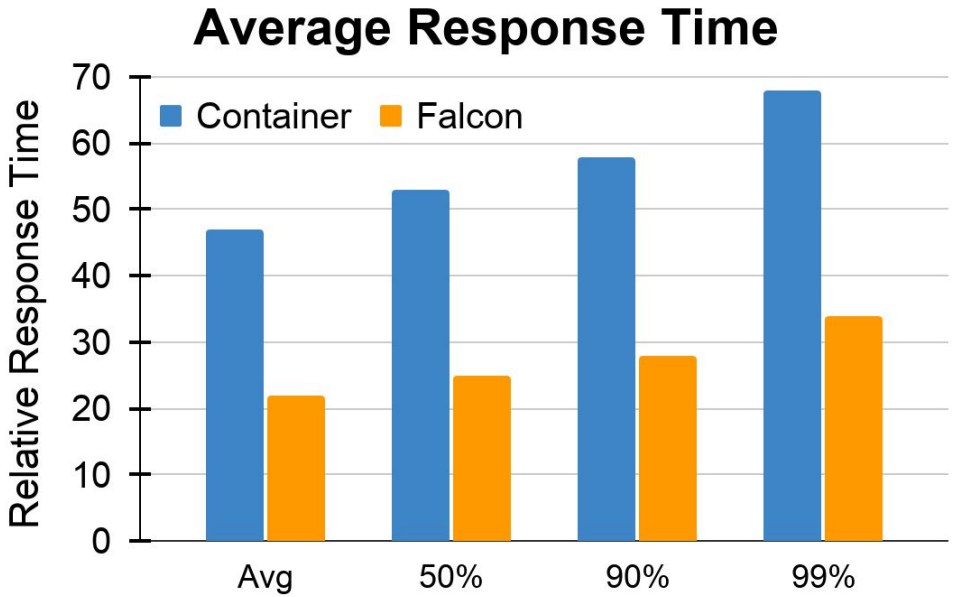
- Throughput improved by up to 300%
- Response time reduced by up to 31%





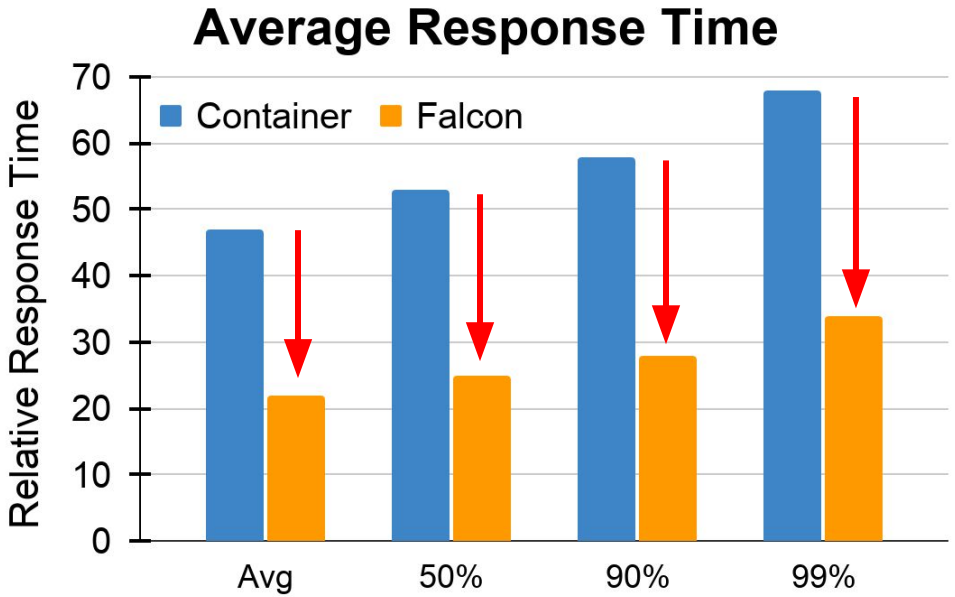
Cloud benchmarks: Data Caching

- Memcached benchmark
  - 4 server threads
  - 10 clients



Cloud benchmarks: Data Caching

- Memcached benchmark
  - 4 server threads
  - 10 clients
- Avg and tail latency reduced to 50%



Conclusion

- Overlay packet processing in current OS is not optimized to utilize multicore
- FALCON accelerates overlay packet processing
  - Without losing any features such as security, flexibility, compatibility
- Purely software-based solution that is easy to deploy and upgrade
- Our implementation is available at github.com/munikarmanish/falcon

Conclusion

- Overlay packet processing in current OS is not optimized to utilize multicore
- FALCON accelerates overlay packet processing
  - Without losing any features such as security, flexibility, compatibility
- Purely software-based solution that is easy to deploy and upgrade
- Our implementation is available at github.com/munikarmanish/falcon

Thank you!