Datacenter RPCs can be General and Fast

Anuj Kalia (CMU)

Michael Kaminsky (Intel Labs)

David G. Andersen (CMU)

Modern datacenter networks are fast

- 100 Gbps
- 2 μs RTT under one switch
- 300 ns per switch hop

Existing networking options sacrifice performance or generality

General Slow	↔	Specialized Fast
Ex: TCP, gRPC		Ex: DPDK, RDMA

- Works in commodity datacenters
- Provides reliability, congestion control, …

- Makes simplifying assumptions
- Requires special hardware

Specialization for fast networking

RDMA NICs

FaRM [NSDI 14, SOSP 15]
HERD [SIGCOMM 14]
DrTM [SOSP15, OSDI 18]
LITE [SOSP 17]
Wukong [OSDI 16]
FaSST [OSDI 16]
NAM-DB [VLDB 17]
HyperLoop [SIGCOMM 18]
DSLR [SIGMOD 18]
…

FPGAs

KV-Direct [SOSP 15]
ZabFPGA [NSDI 18]

Programmable switches

NetChain [NSDI 18]

Drawbacks

- Limited applicability
- Reduced modularity and reuse due to co-design

eRPC provides both speed and generality

General ⟷ Specialized

Slow ⟷ Fast

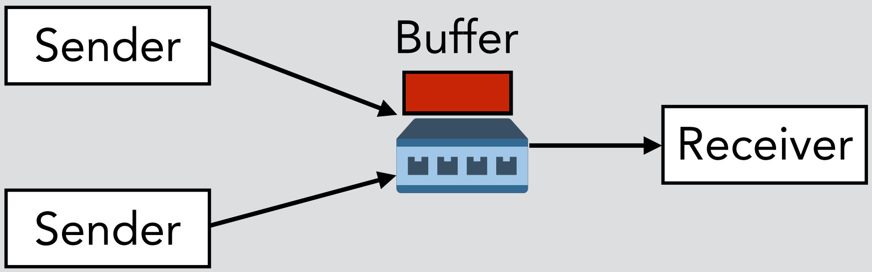
- Works in commodity datacenters
- Provides reliability, congestion control, …

Three challenges

1. Managing packet loss
2. Low-overhead transport
3. Easy integration for existing applications

Challenge #1: Managing packet loss

Problem: Millisecond timeouts for small RPCs

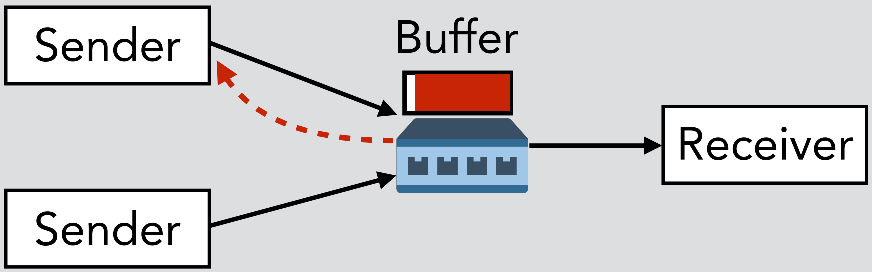


If a client's unlock packet is dropped:

- Client retransmits after many **milliseconds**
- Many contending requests fail

Challenge #1: Managing packet loss

Problem: Millisecond timeouts for small RPCs



If a client's unlock packet is dropped:

- Client retransmits after many **milliseconds**
- Many contending requests fail

Hardware solution: Lossless link layer (e.g., PFC, InfiniBand)

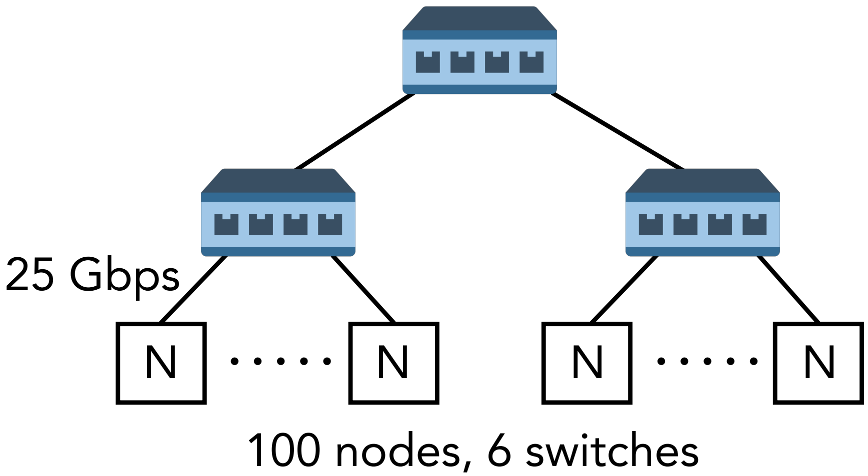
Pros: Simple/cheap reliability

Cons: Deadlocks, unfairness

eRPC's solution

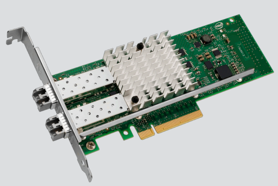
A relaxed requirement for rare loss, supported by existing networks

In low-latency networks, switch buffers prevent most loss

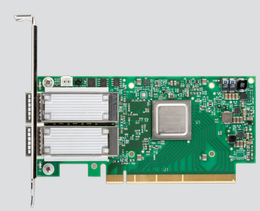


- Bandwidth = 25 Gbps, RTT = 6.0 µs
- Bandwidth x delay (BDP) = 19 KB
- Switch buffer = 12 MB >> BDP

Enabled by low-latency NICs



Slow NIC
Adds 10 µs



Fast NIC
Adds 500 ns

All modern switches have buffers >> BDP

Broadcom Trident 3 (32 MB)

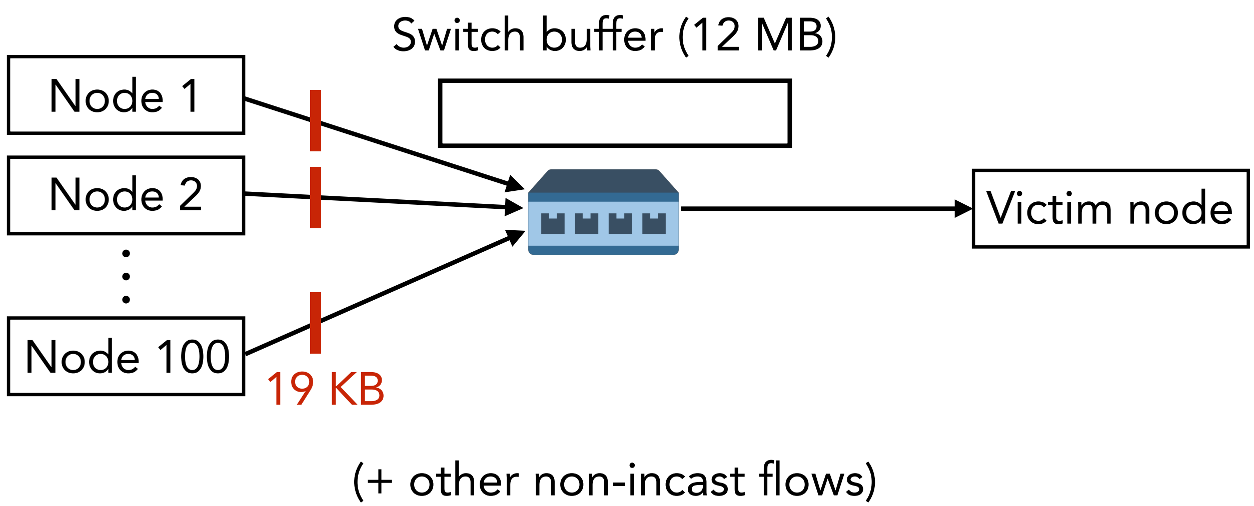
Mellanox Spectrum 2 (42 MB)

Barefoot Tofino (22 MB)

These are not "big buffer" switches!

Cisco 3636-C (16 **giga**bytes, DRAM buffer)

Small BDP + sufficient switch buffer ⇒ Rare loss



- Incast tolerance = 12 MB / 19 KB = 640
  ≈ 50-way tolerance desired in practice [e.g., DCQCN @Microsoft, Timely @Google]
- Tested with 100-way incast: No loss

Challenge #2: Low-overhead transport layer

Idea: Optimize for the common case

Example 1: Optimized DMA buffer management for rare packet loss

Example 2: Optimized congestion control for uncongested networks

Many more in paper:

- Optimized memory allocation for small-size RPCs
- Optimized threading for short-duration RPCs
- …

Example: Optimized DMA buffer management for rare packet loss

Problem: Detecting completion of request DMA

Method #1: Explicit NIC signal

- Overhead for each request

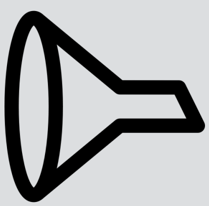
Method #2: Server’s response

- Free
- Doesn’t work if a packet is lost

Solution: Use server’s response in common case. Flush DMA queue during rare loss.

Example: Efficient congestion control in software

Problem: Congestion control overhead



Example: Rate limiter overhead

Hardware solution: NIC offload

Pro: Saves CPU cycles

Con: Low flexibility

Ex: Difficult to use Carousel [SIGCOMM 17]

eRPC's solution

Optimize for uncongested networks

Datacenter networks are usually uncongested

Facebook datacenter studies

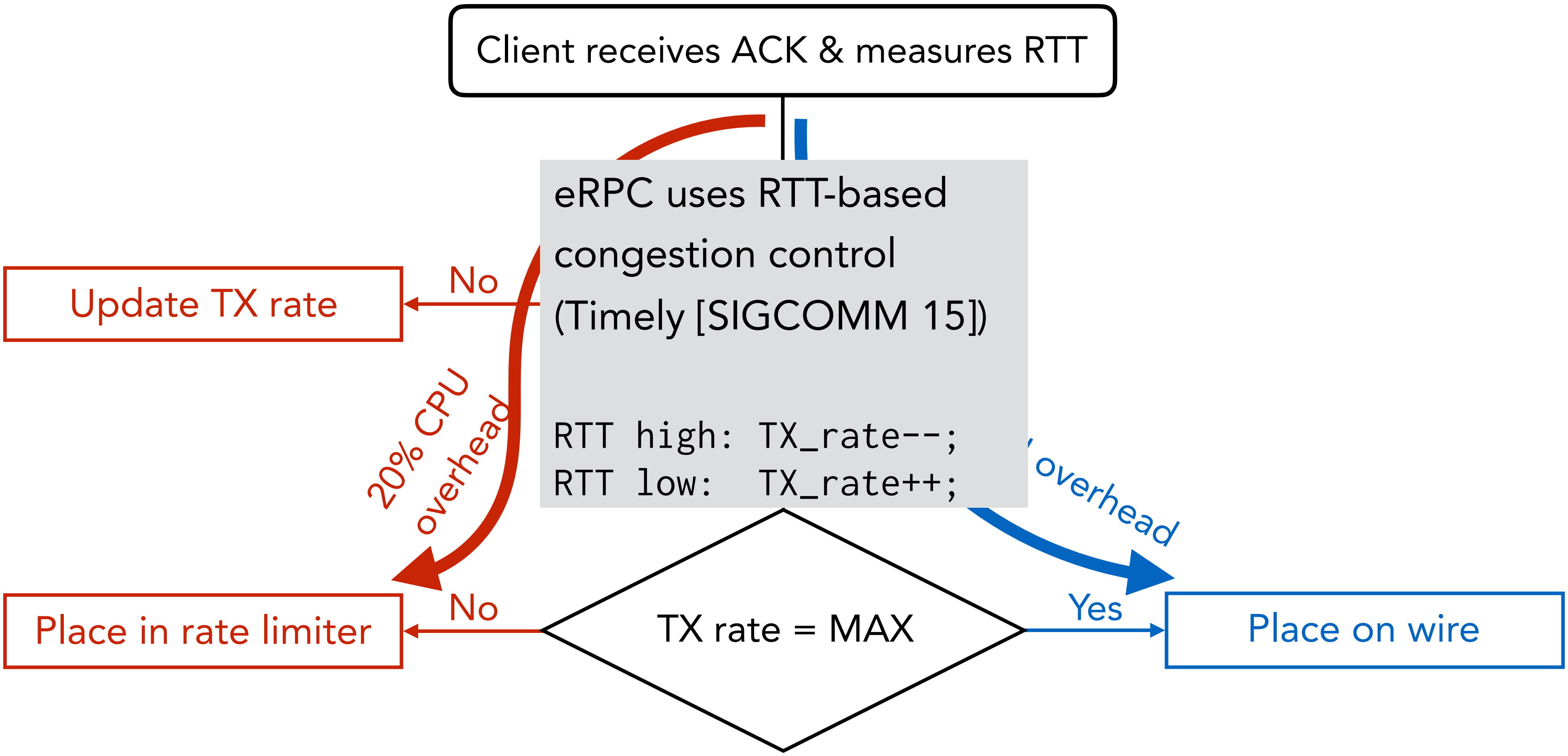
Timescale	Links less than 10% utilized
Ten minutes	99% [Roy et al., SIGCOMM 15]
25 μs	90% [Zhang et al., IMC 17]

Congestion control, fast and slow

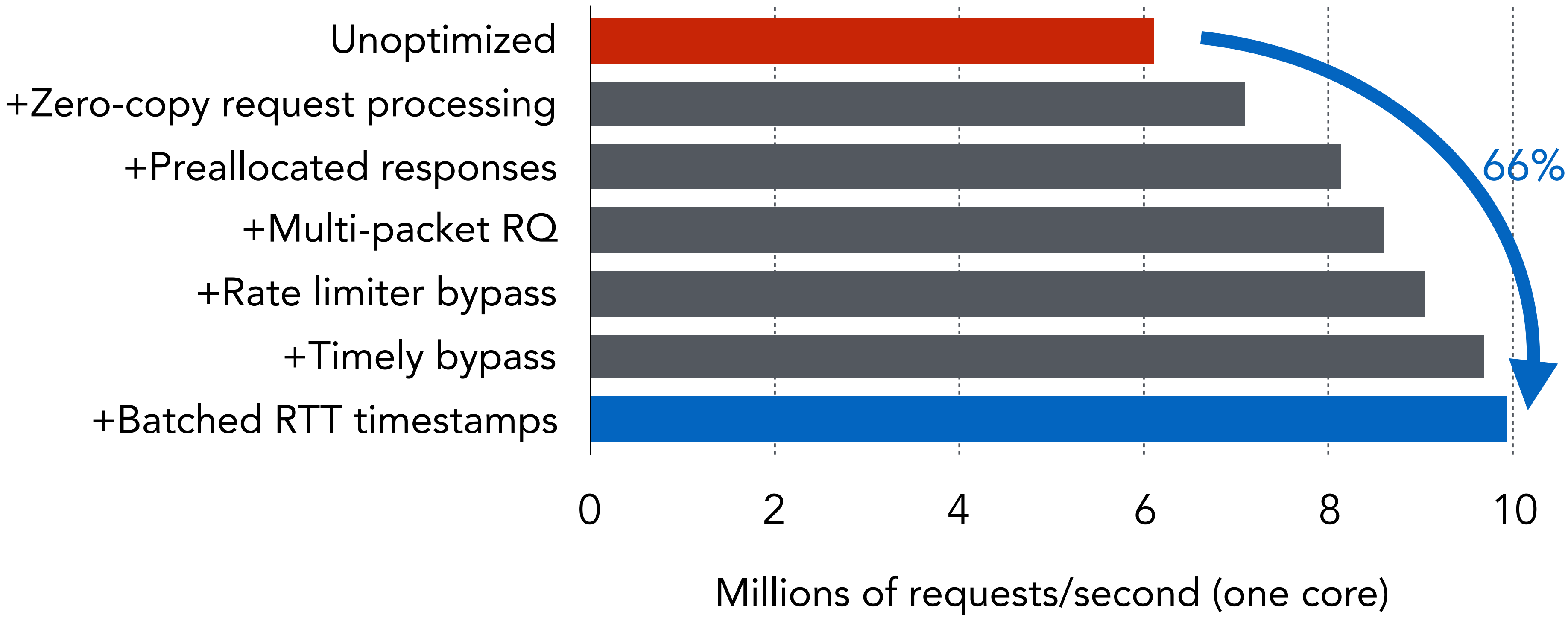
eRPC uses RTT-based congestion control (Timely [SIGCOMM 15])

```
RTT high: TX_rate--;
RTT low:  TX_rate++;
```

Congestion control, fast and slow



Together, common-case optimizations matter



Result: Low overhead transport *with* congestion control

eRPC microbenchmark highlights

Lossy 40 GbE network

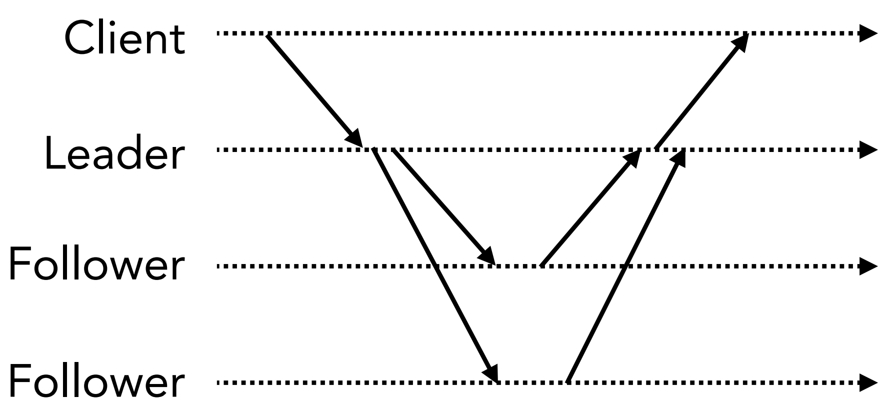
- 2.3 µs RPC round-trip latency
- Line rate with one core
- 60 million RPCs/s per machine
- Scalability to 20000 connections (>> RDMA)

Challenge #3: Easy integration with existing applications

willemt / raft

- 5 years of developer effort. 150+ unit tests, fuzzing.
- In production use by Intel

Remote procedure calls in Raft



Complexity during failure

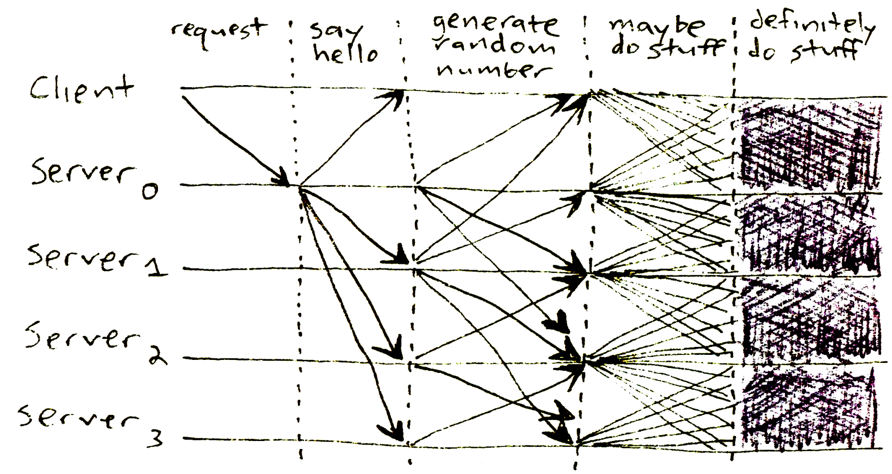
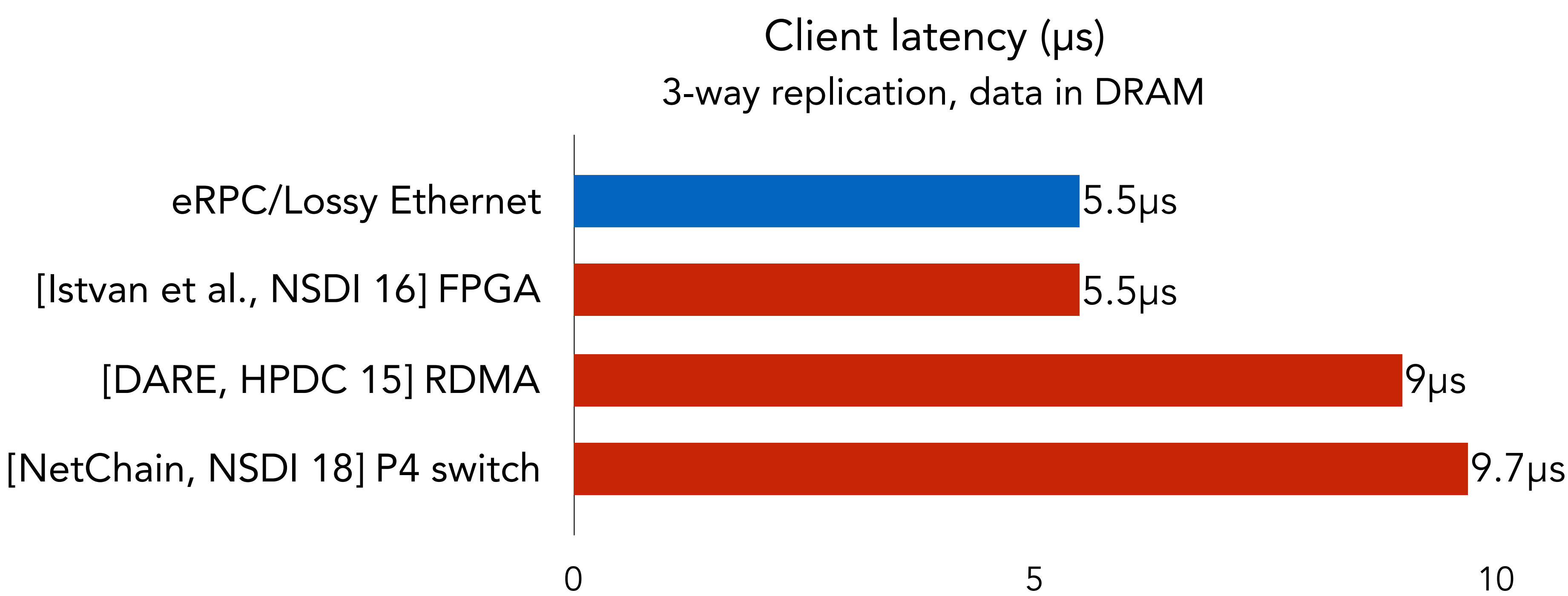


Image credit: James Mickens

Replication over eRPC is fast



Raft-over-eRPC does not have network or object size constraints

Takeaway: Given fast packet I/O, we can provide fast networking in software

"Using performance to justify placing functions in a low-level subsystem must be done carefully.

Sometimes, by examining the problem thoroughly, the same or better performance can be achieved at the high level."

— End-to-end Arguments in System Design [Saltzer, 84]

erpc.io

Industry impact: https://github.com/daq-db/

I am on the academic job market