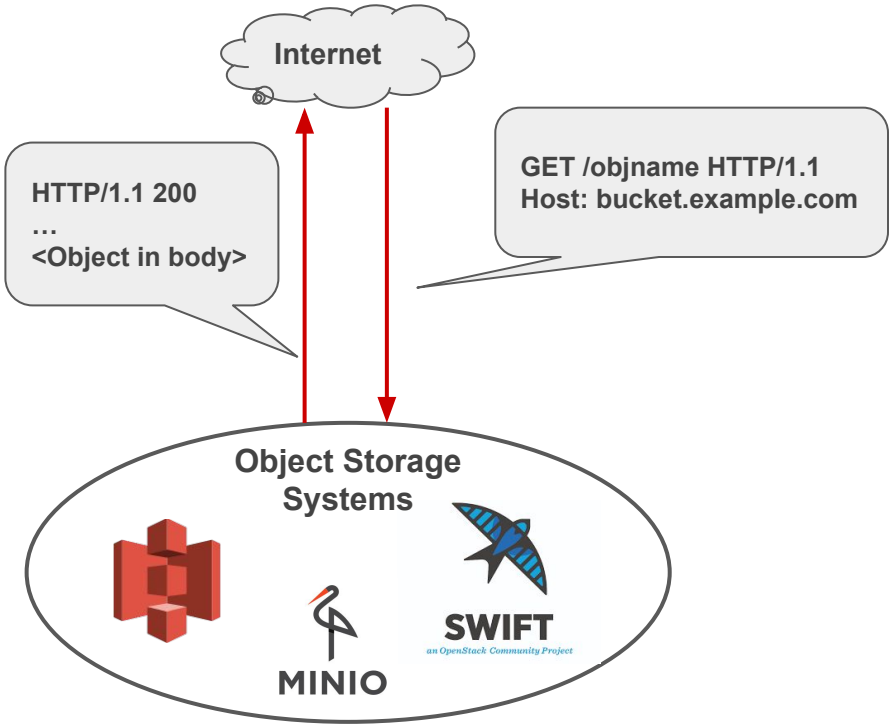
Prism: Proxies without the Pain

Yutaro Hayakawa (LINE Corporation)
Michio Honda (University of Edinburgh)
Douglas Santry (Apple Inc.)
Lars Eggert (NetApp)

Object Storage Systems



Simple HTTP based flat namespace storage service

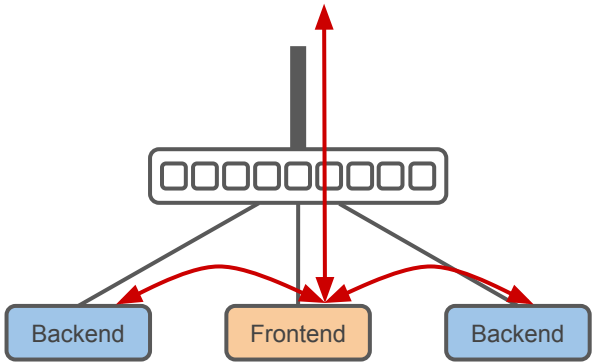
Amazon S3, Google Cloud Storage, etc…

Use cases

Video streaming, container image registry, etc...

Object Storage Scale-out Options

Frontend Proxy



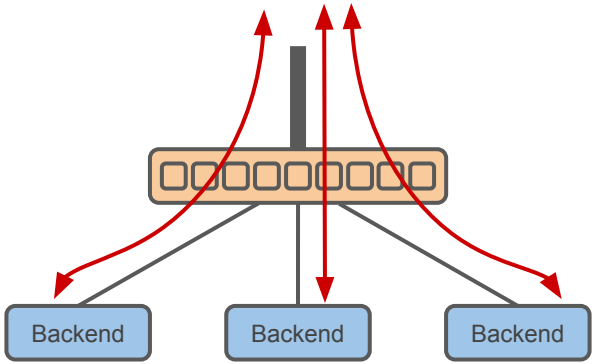
OpenStack Swift, Minio, Mos [HPDC'16], Crystal [FAST'17], etc...

✅ Standard protocols (e.g. S3 / HTTP)

✅ TLS (Encryption / Authentication)

❌ Frontend becomes bottleneck

Content-based Routing



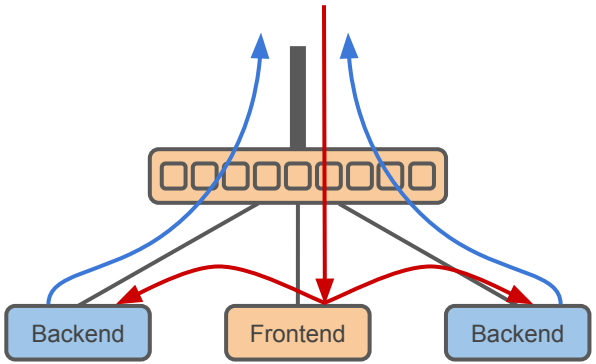
SwitchKV [NSDI'16], NetCache [SOSP'17], Pegasus [OSDI'20], etc...

✅ No intermediate servers

❌ Requires special protocols in client-side

❌ Cannot use TLS

Prism



Our proposal

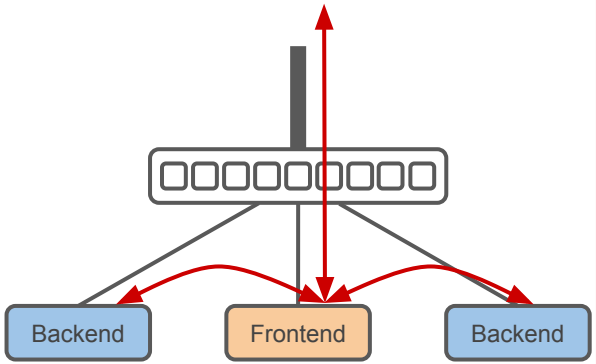
✅ Can use standard protocols

✅ TLS

✅ No intermediate servers (for return path)

Object Storage Scale-out Options

Frontend Proxy



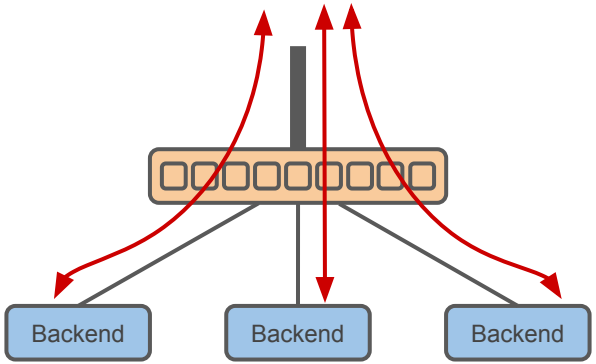
OpenStack Swift, Minio, Mos [HPDC'16], Crystal [FAST'17], etc...

✅ Standard protocols (e.g. S3 / HTTP)

✅ TLS (Encryption / Authentication)

❌ Frontend becomes bottleneck

Content-based Routing



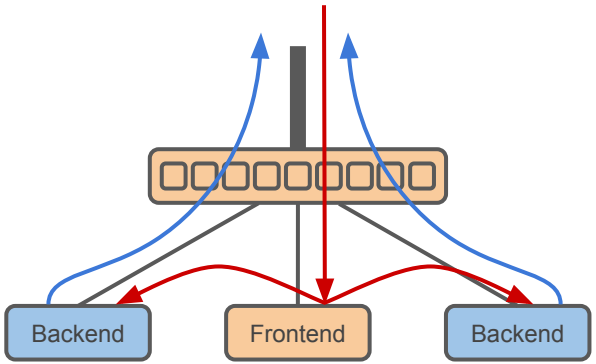
SwitchKV [NSDI'16], NetCache [SOSP'17], Pegasus [OSDI'20], etc...

✅ No intermediate servers

❌ Requires special protocols in client-side

❌ Cannot use TLS

Prism



Our proposal

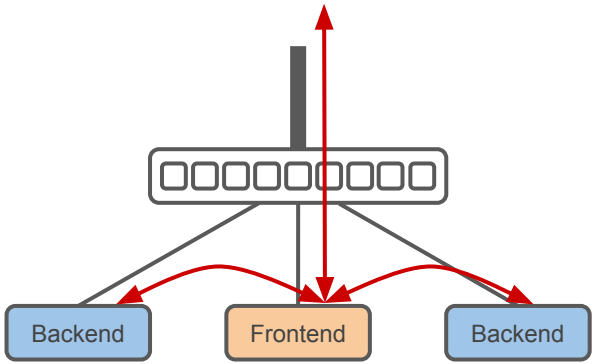
✅ Can use standard protocols

✅ TLS

✅ No intermediate servers (for return path)

Object Storage Scale-out Options

Frontend Proxy



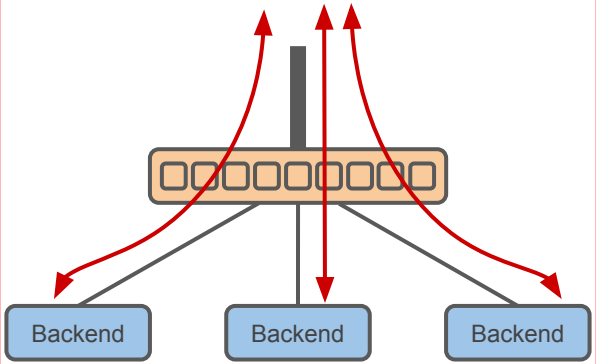
OpenStack Swift, Minio, Mos [HPDC'16], Crystal [FAST'17], etc...

✔ Standard protocols (e.g. S3 / HTTP)

✔ TLS (Encryption / Authentication)

✘ Frontend becomes bottleneck

Content-based Routing



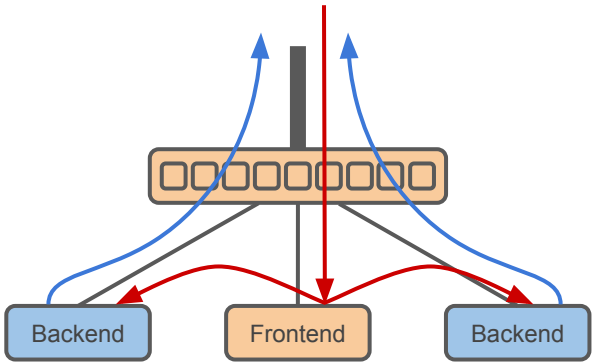
SwitchKV [NSDI'16], NetCache [SOSP'17], Pegasus [OSDI'20], etc...

✔ No intermediate servers

✘ Requires special protocols in client-side

✘ Cannot use TLS

Prism



Our proposal

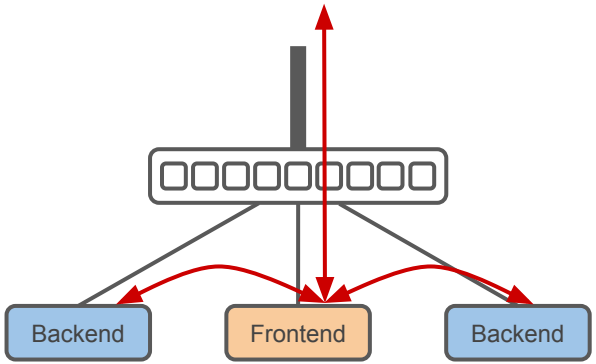
✔ Can use standard protocols

✔ TLS

✔ No intermediate servers (for return path)

Object Storage Scale-out Options

Frontend Proxy



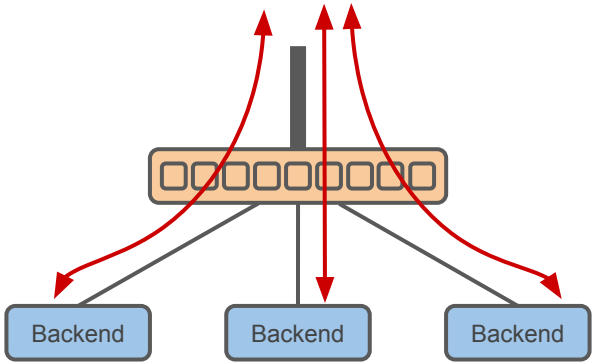
OpenStack Swift, Minio, Mos [HPDC'16], Crystal [FAST'17], etc...

✅ Standard protocols (e.g. S3 / HTTP)

✅ TLS (Encryption / Authentication)

❌ Frontend becomes bottleneck

Content-based Routing



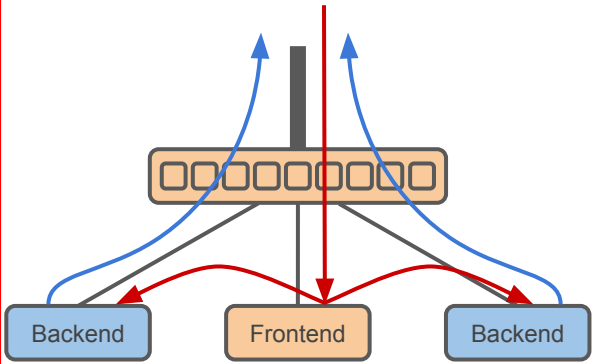
SwitchKV [NSDI'16], NetCache [SOSP'17], Pegasus [OSDI'20], etc...

✅ No intermediate servers

❌ Requires special protocols in client-side

❌ Cannot use TLS

Prism



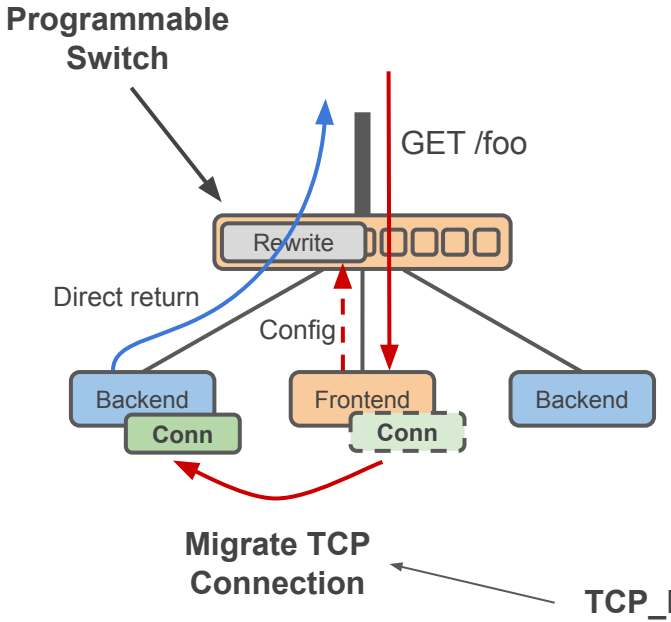
Our proposal

✅ Can use standard protocols

✅ TLS

✅ No intermediate servers (for return path)

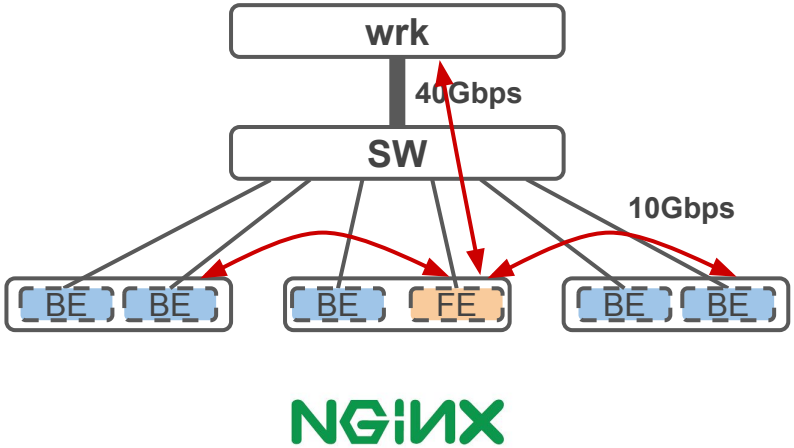
TCP Hand-off [ASPLOS'98]



Now it is more feasible than as of 1998

1. Linux TCP_REPAIR
2. Programmable DPlane eco-system (P4, eBPF...)

Performance Characterization of Frontend Proxy Architecture



Benchmark

1. S3 protocol with HTTP(S) keep-alive
2. 100 parallel TCP connections
3. Round-robin load balancing on frontend
4. Fetches in-memory objects

Performance Characterization of Frontend Proxy Architecture

Left Y-Axis: Cluster CPU Utilization

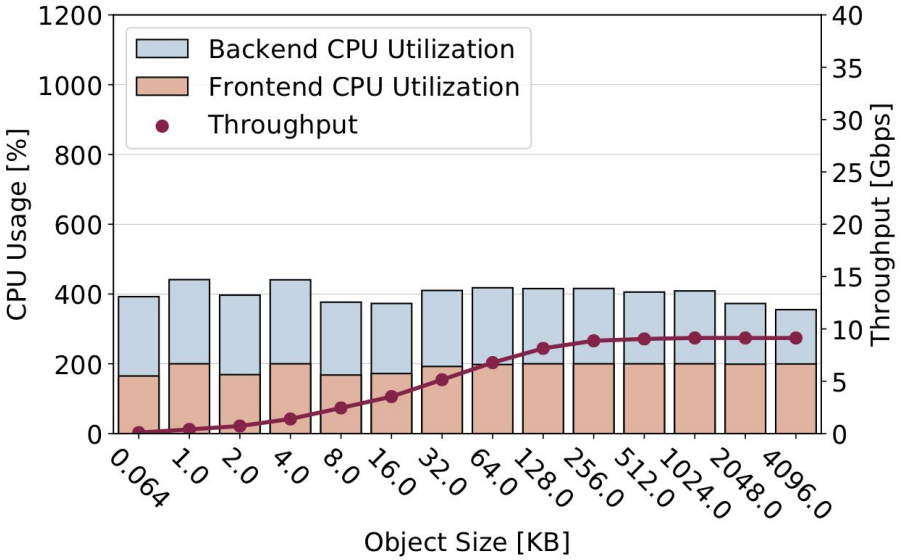
1200% = 2cores (200%) * 5 backends
+ 2cores (200%) * 1 frontend

Right Y-Axis: Throughput observed at client

Max 40Gbps (attached bandwidth of the client)

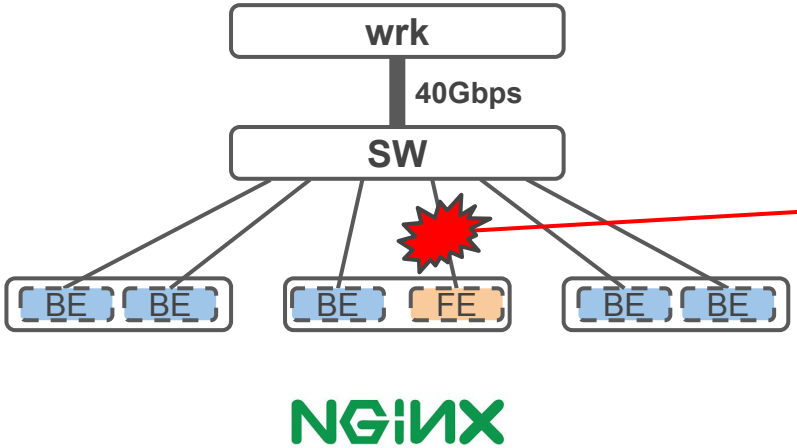
X-Axis: Object size with KB

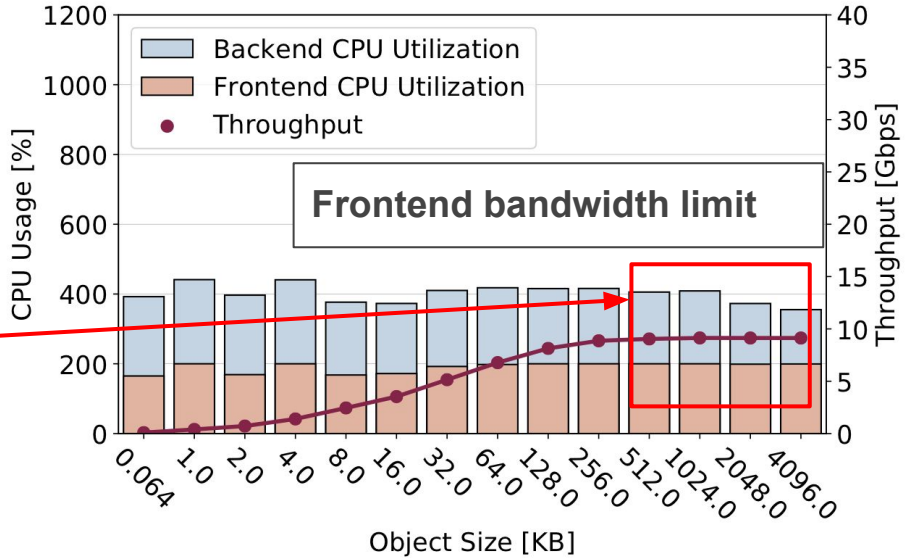
0.064 => 64bytes



Underutilized Uplink Bandwidth

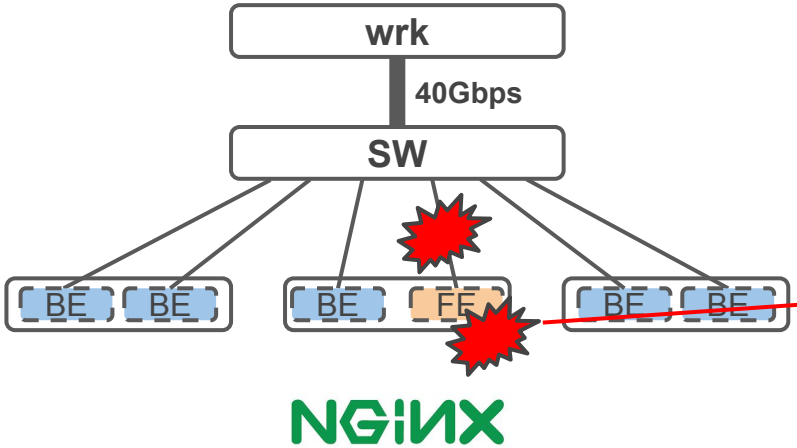
30Gbps out of 40Gbps uplink bandwidth is unused

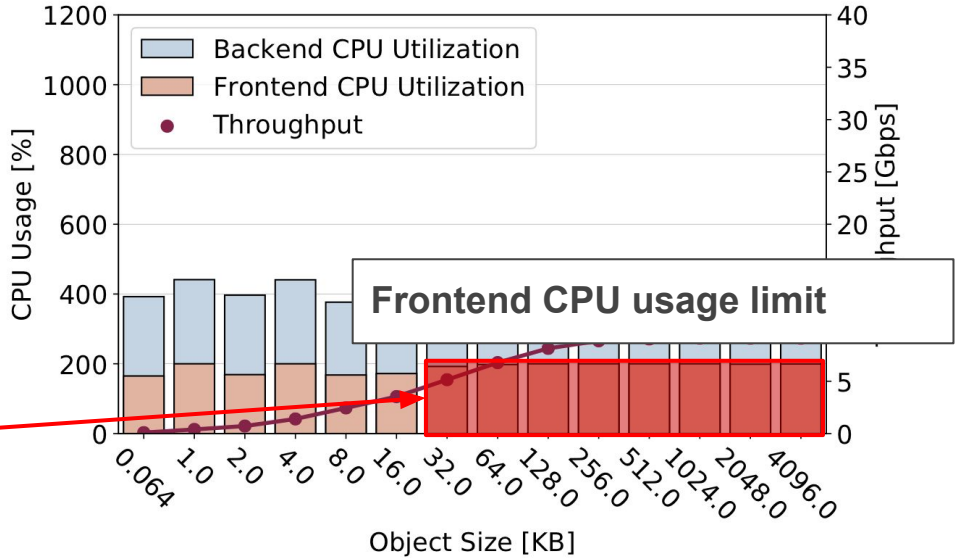




Overutilized Frontend CPU Resources

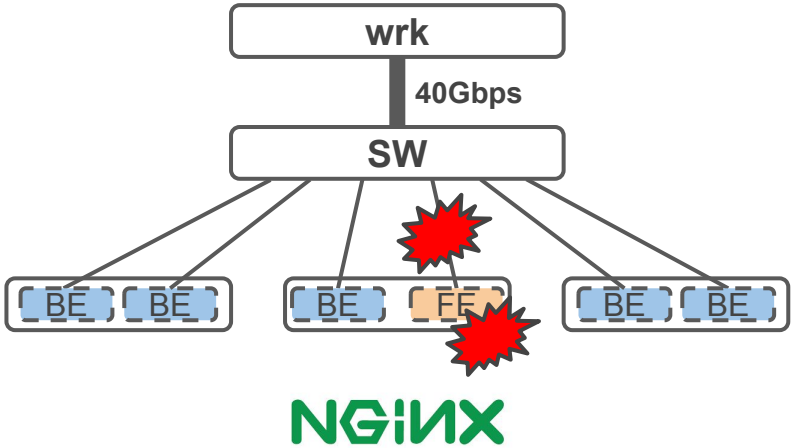
Frontend CPU is fully utilized in most of the cases

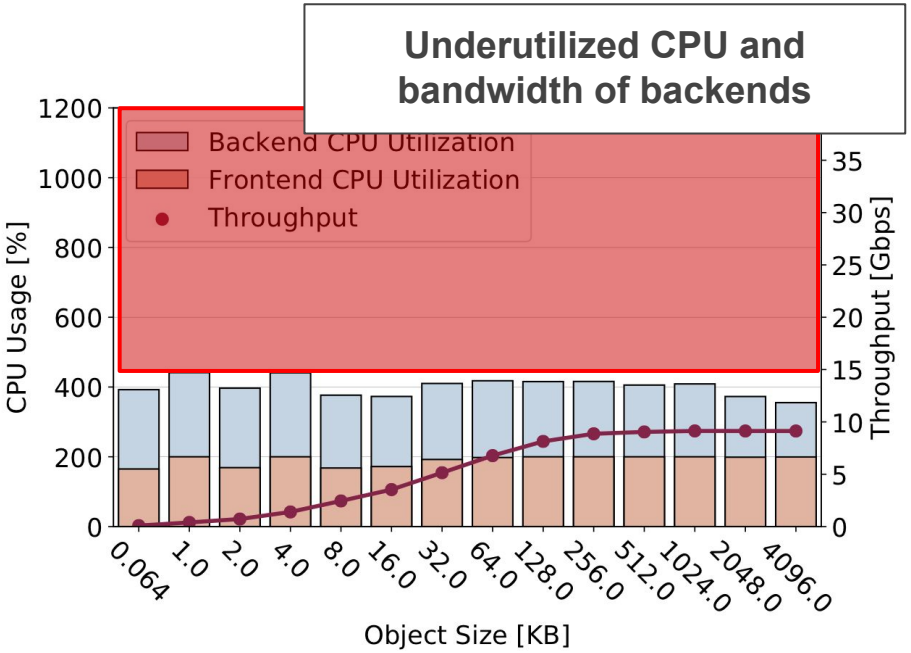




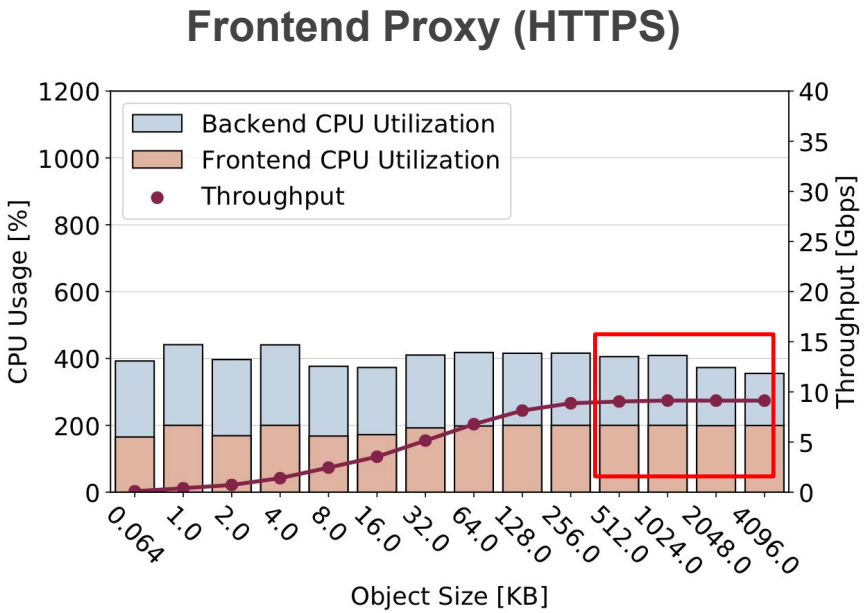
Underutilized Backend CPU Resources

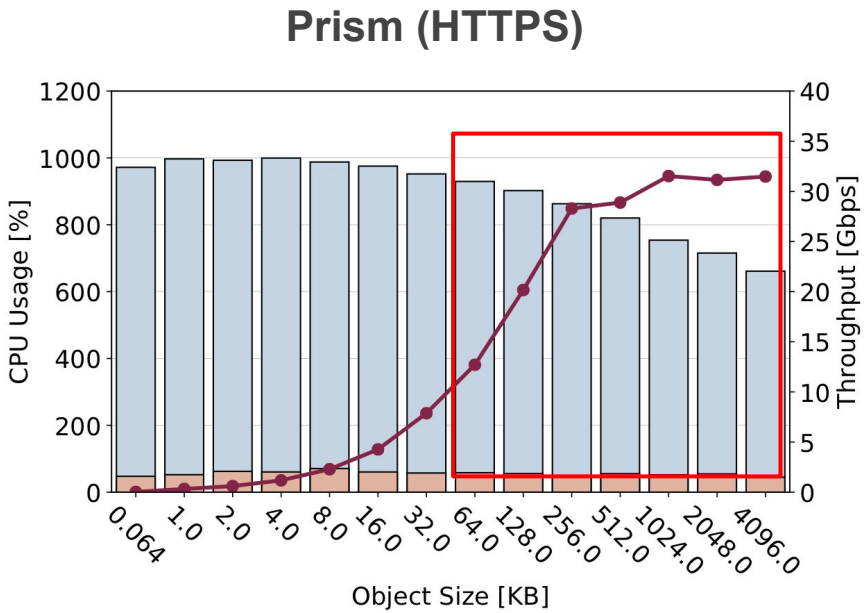
Backend resources are underutilized



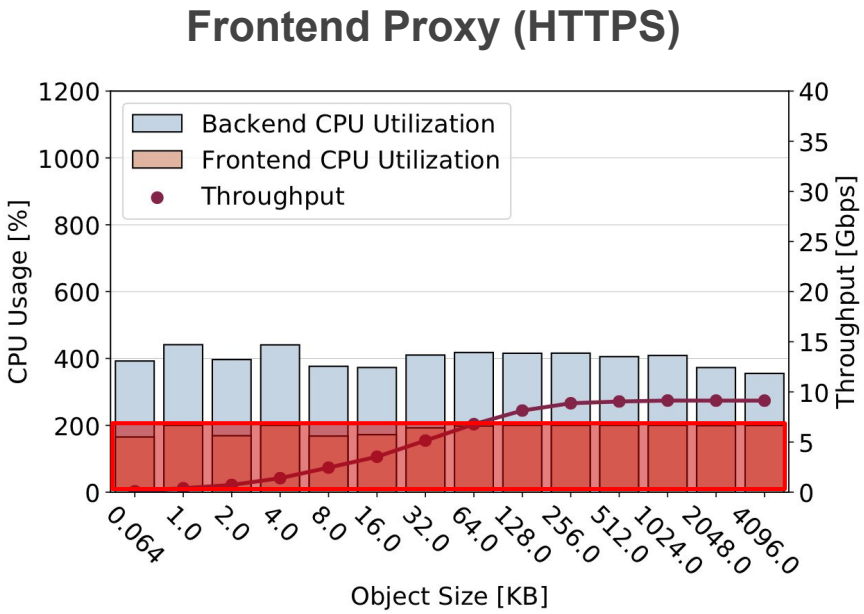


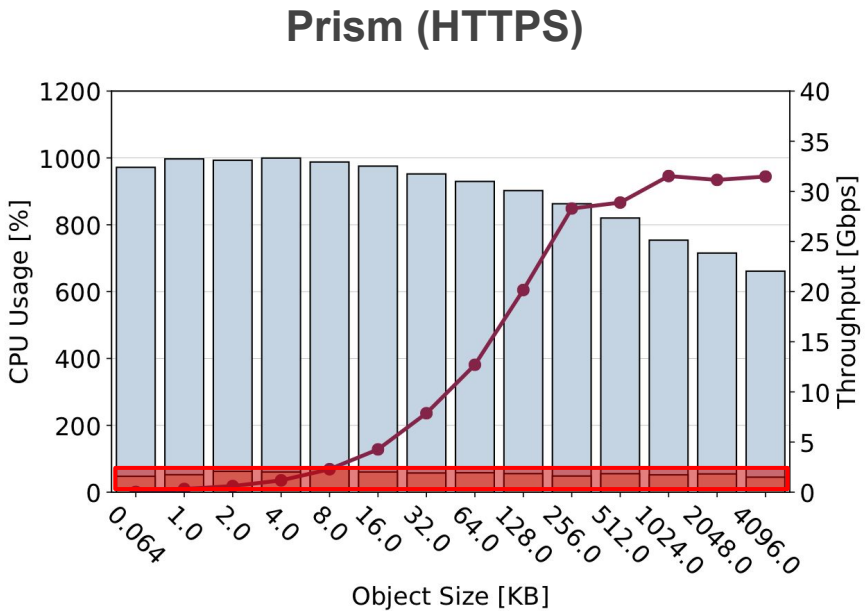
Prism Effectively Utilizes Uplink Bandwidth



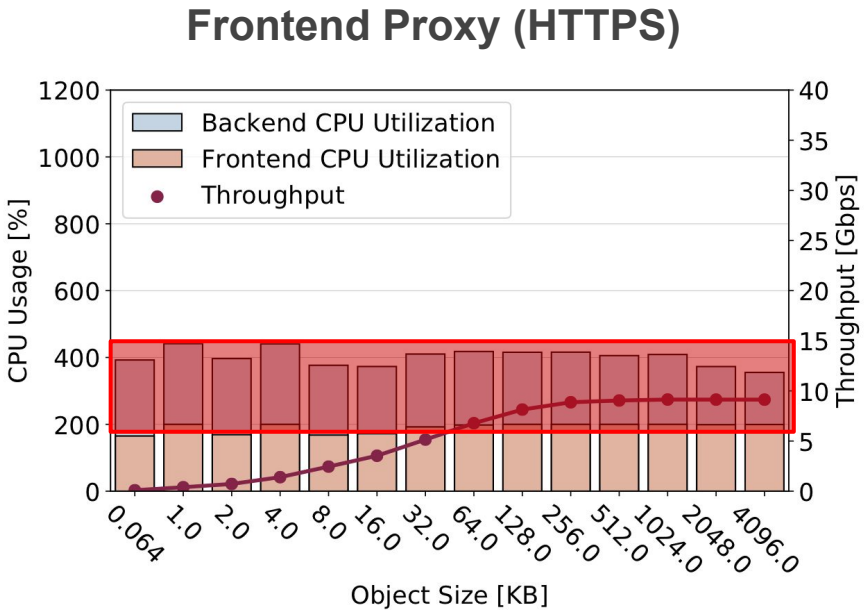


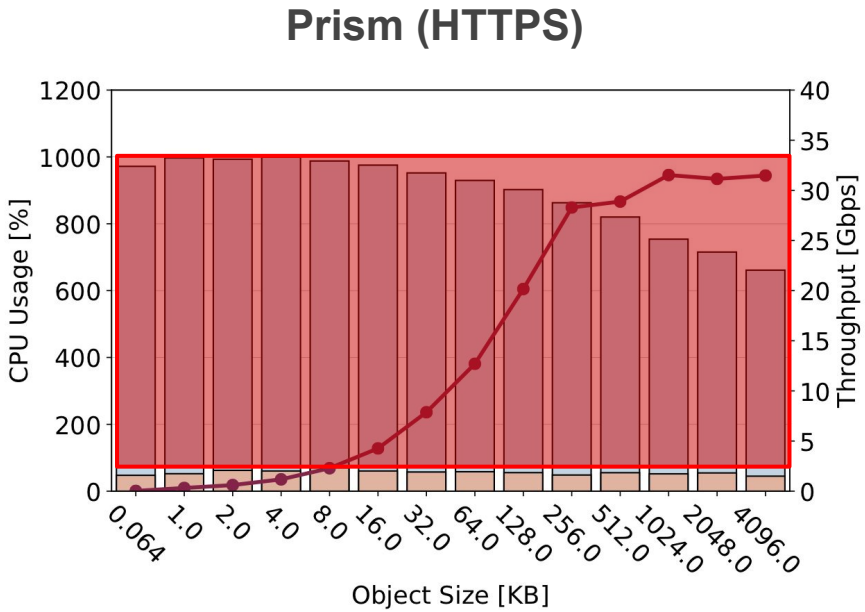
Prism Effectively Reduces Frontend CPU Utilization





Prism Effectively Utilizes Backend CPU Resources





Our Contributions

1. Robust hand-off protocol

Two Phase Hand-off Protocol

2. eBPF-based switch logic

Efficient switch interaction / inline switch configuration protocol

3. Stack with APIs, switch interaction and kernel module

Event loop based software stack to integrate TCP Hand-off to the applications

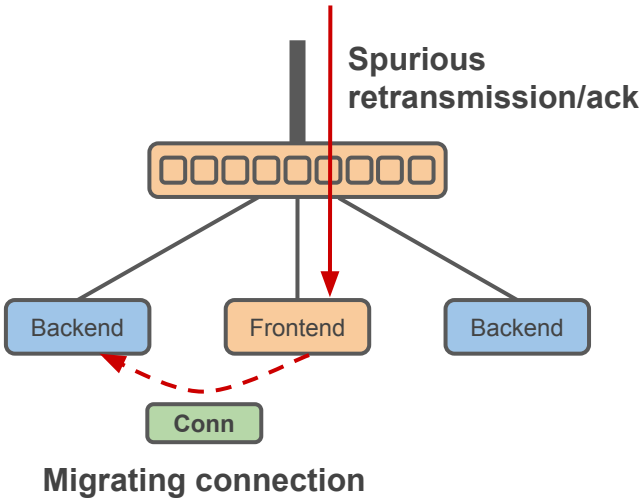
FIN_WAIT1/CLOSE_WAIT handling

4. Evaluation with realistic use cases

Replicated / Partitioned object storage, measurement with YCSB workload

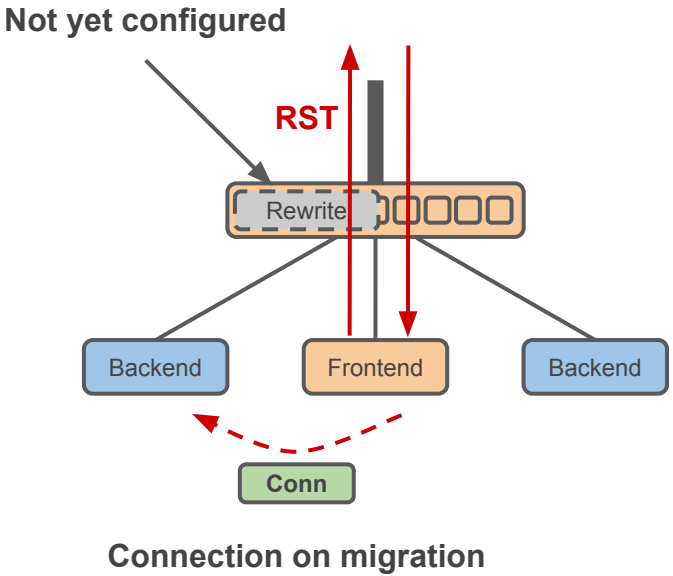
Details in the paper

TCP Hand-off is Non-trivial



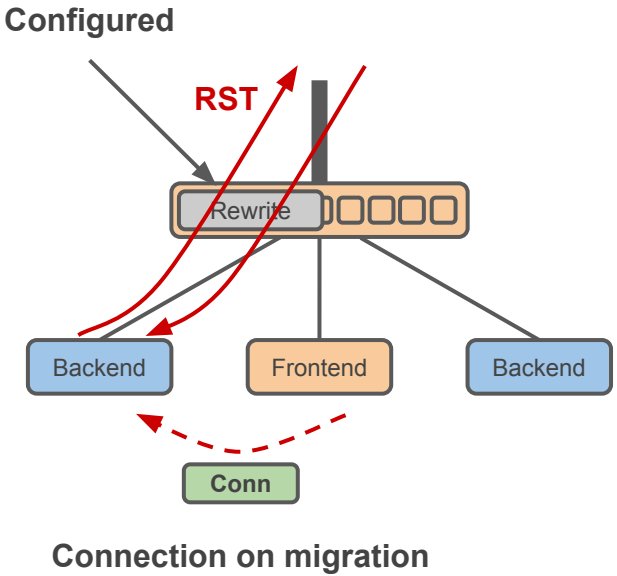
Which server should receive unexpected packets?

TCP Hand-off is Non-trivial



“Old server” sends a reset when the connection has gone

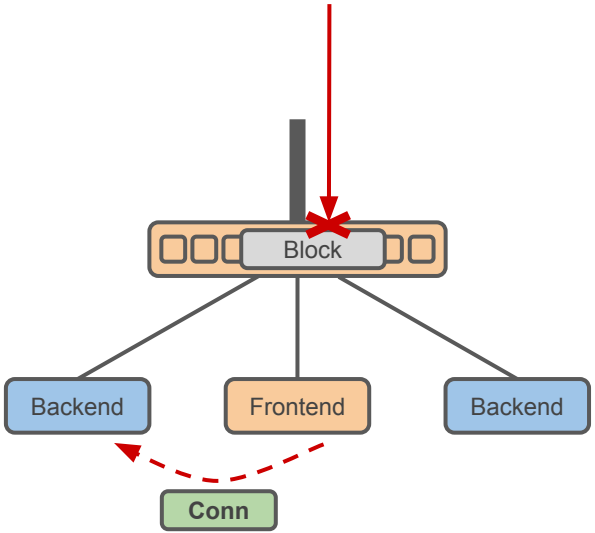
TCP Hand-off is Non-trivial



Connection on migration

“New server” sends a reset when the connection hasn’t arrived yet

Two-phase Hand-off



Block client packet before migration to prevent the "leak"

Add one extra step to hand-off procedure

Hand-off Latency

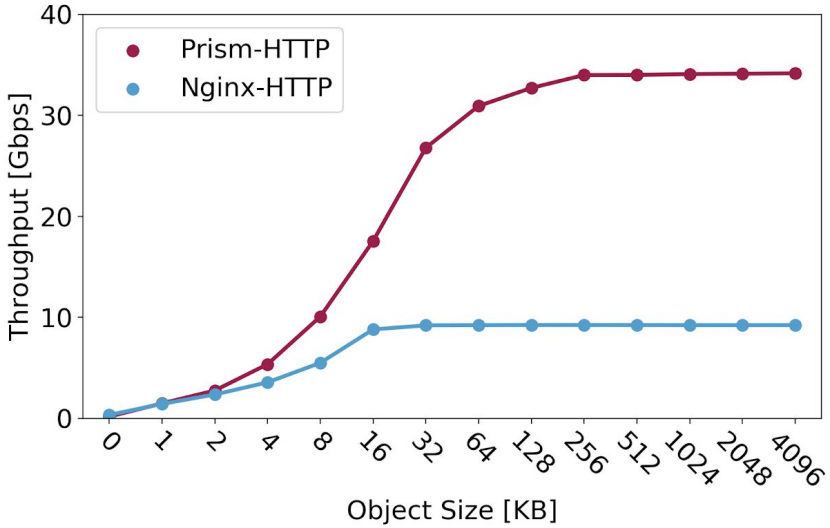
Operation	Latency (μs)
Block all traffic (switch config)	22
Serialize protocol states	14
Send states	51
Deserialize protocol states	123
Configure rewrite (switch config)	22
Total	**232**

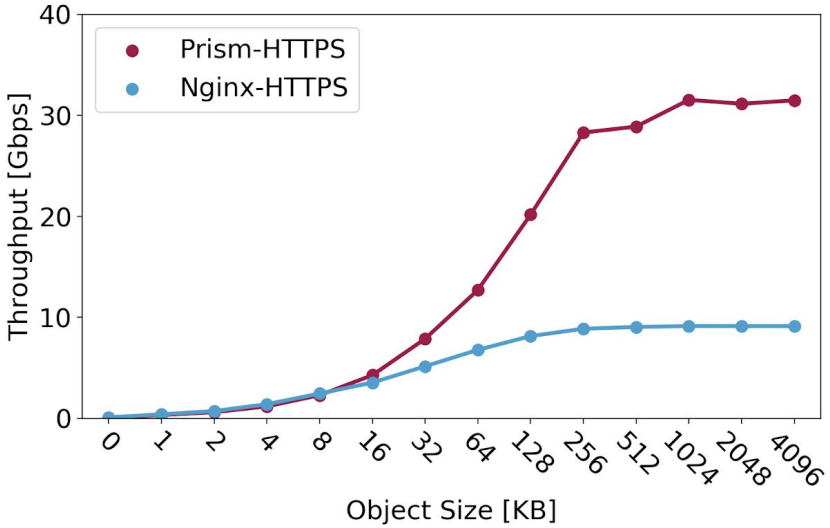
Time takes for hand-off is **fixed** for read

How it costs depends on the **object size**

Cost of Hand-off Latency

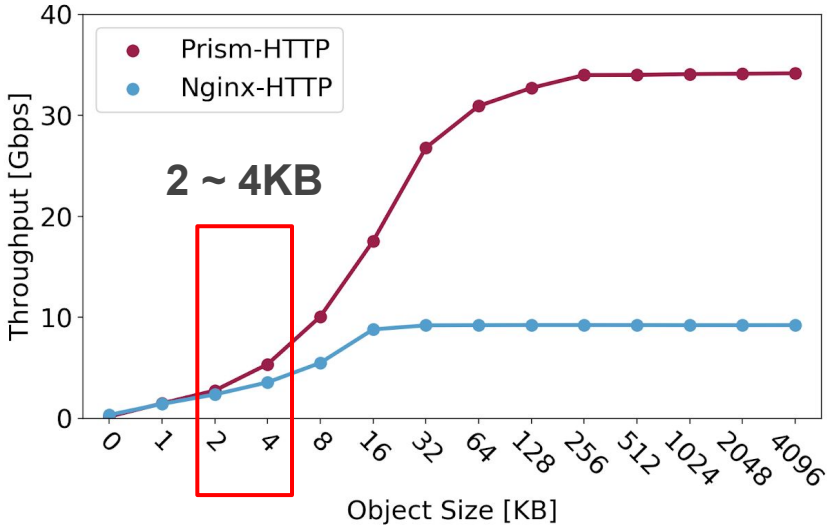
How large does the object size need to be to amortize the connection hand-off cost?

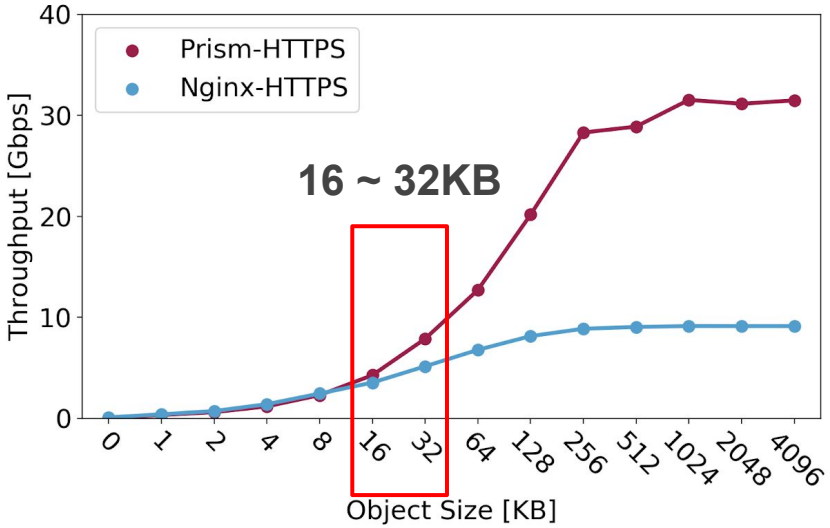




Cost of Hand-off Latency

How large does the object size need to be to amortize the connection hand-off cost?





Recap

Fast storage backends stress object storage frontends

- Attachment link bandwidth and CPU usage

TCP hand-off is promising, but non-trivial

- Promising because of the modern Linux kernel features
- Non-trivial because of the difficulty of coordination with the programmable switch

Prism enables robust TCP hand-off and improves the performance of scale-out systems

Source code is available (no kernel modification needed):
https://github.com/YutaroHayakawa/Prism-HTTP

Thank you