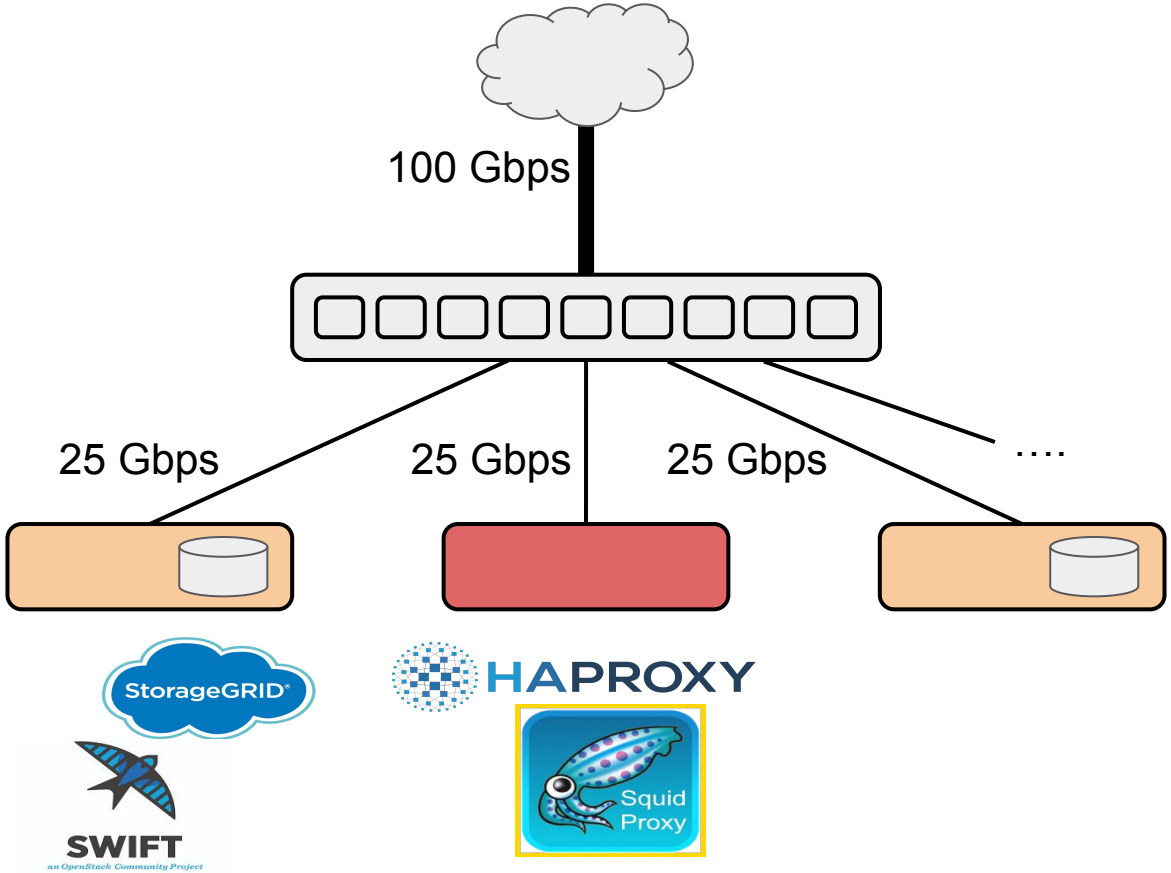
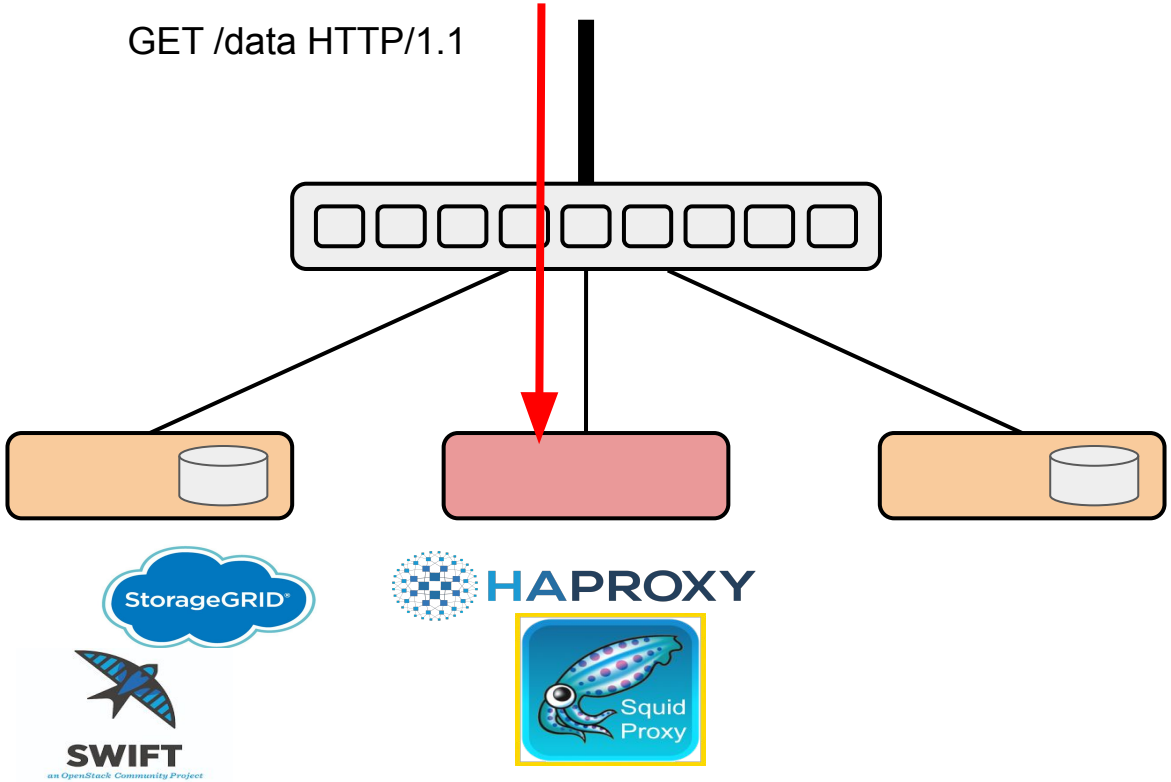
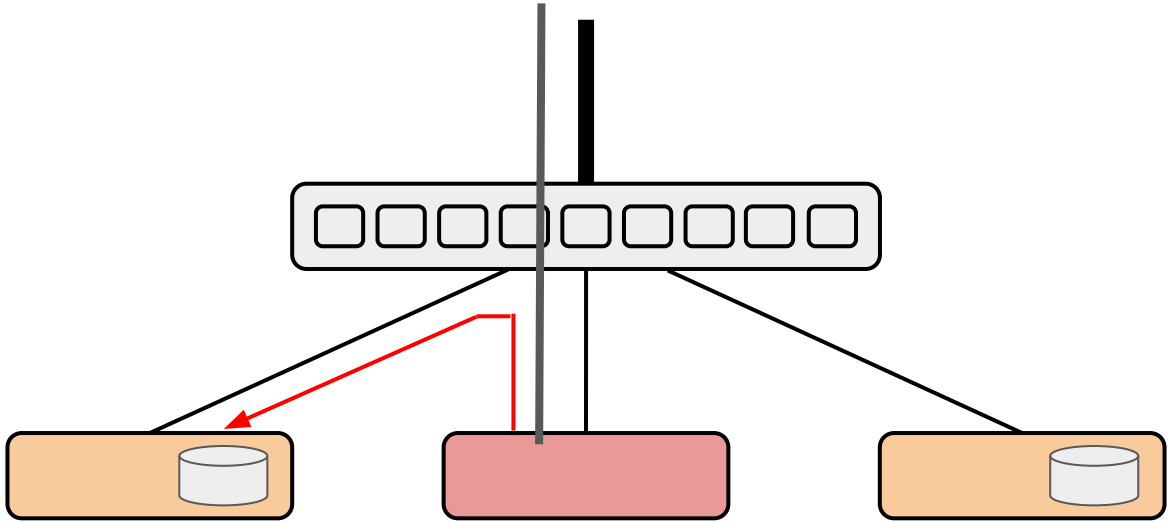
# Remote TCP Connection Offload

Steven W. D. Chien*, Shuo Li*, Tianyi Gao, Michio Honda
University of Edinburgh

NetDev 0x19, Zagreb


100 Gbps
25 Gbps
25 Gbps
25 Gbps
....
StorageGRID
SWIFT
an OpenStack Community Project
HAPROXY
Squid
Proxy


GET /data HTTP/1.1
StorageGRID
SWIFT
an OpenStack Community Project
HAPROXY
Squid
Proxy

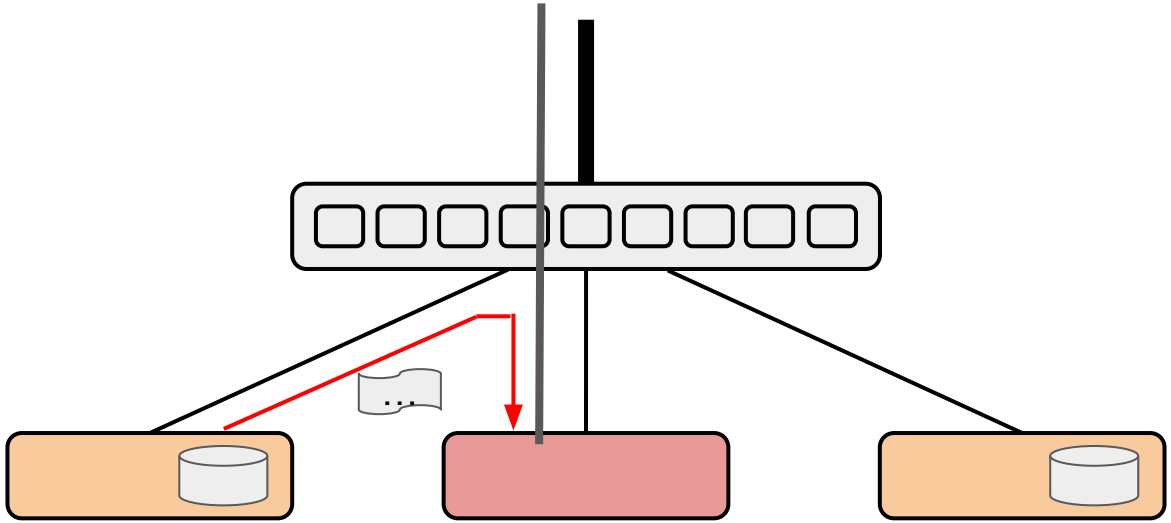


StorageGRID

SWIFT
an OpenStack Community Project

HAPROXY

Squid Proxy

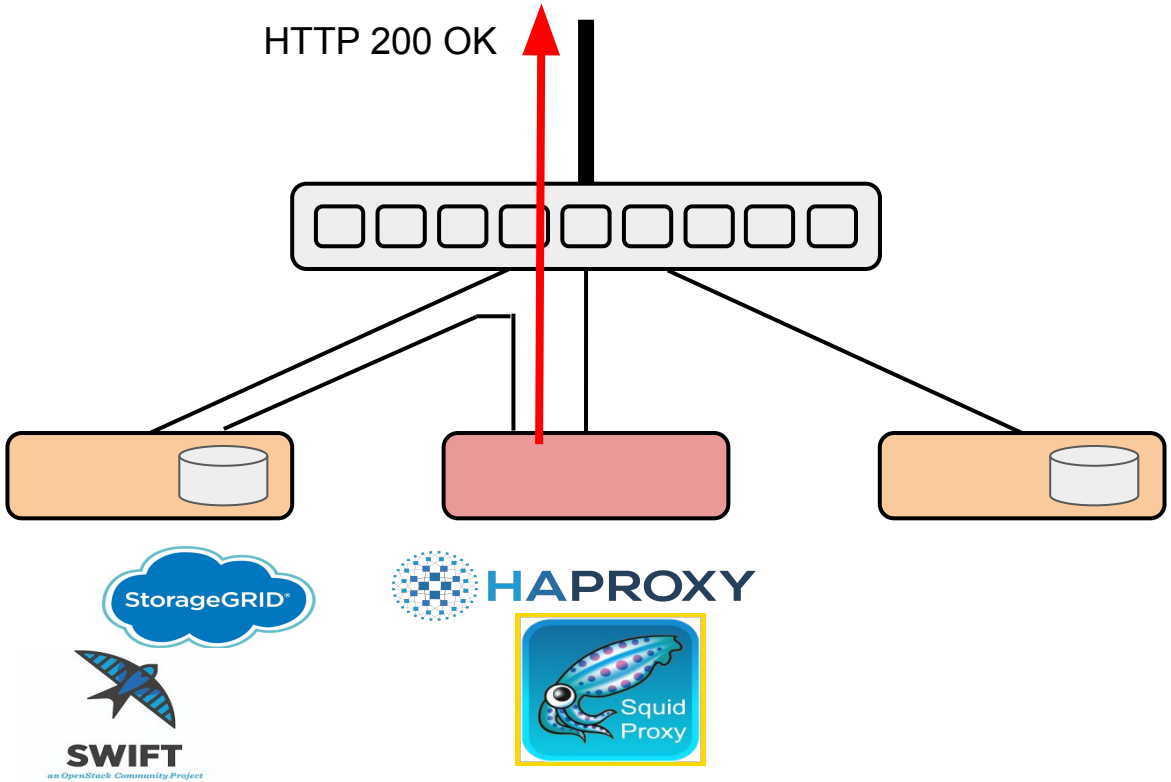


StorageGRID

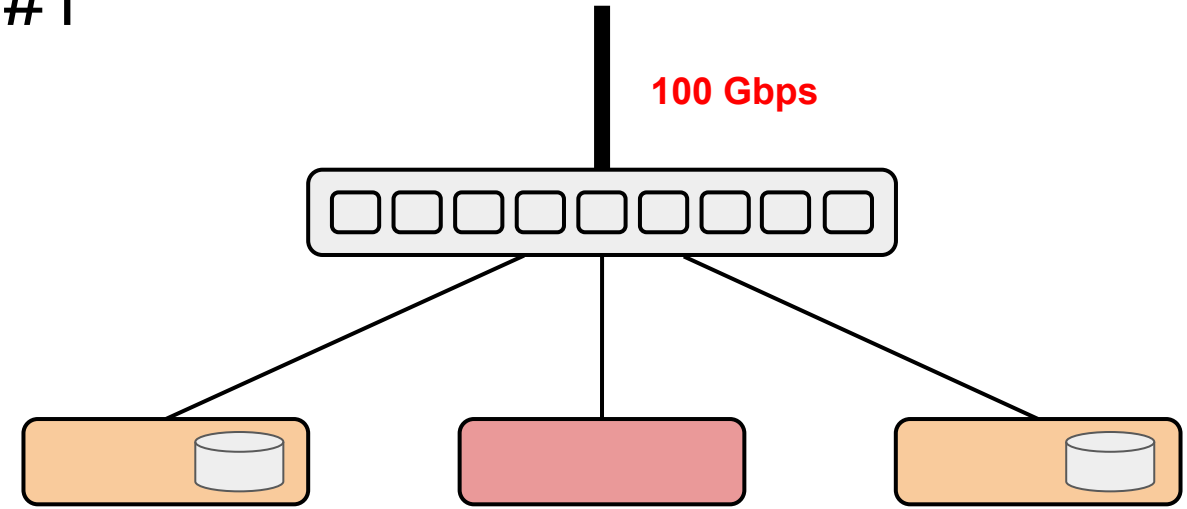
SWIFT
an OpenStack Community Project

HAPROXY

Squid Proxy


HTTP 200 OK
StorageGRID
HAPROXY
Squid
Proxy
SWIFT
an OpenStack Community Project

# Problem #1



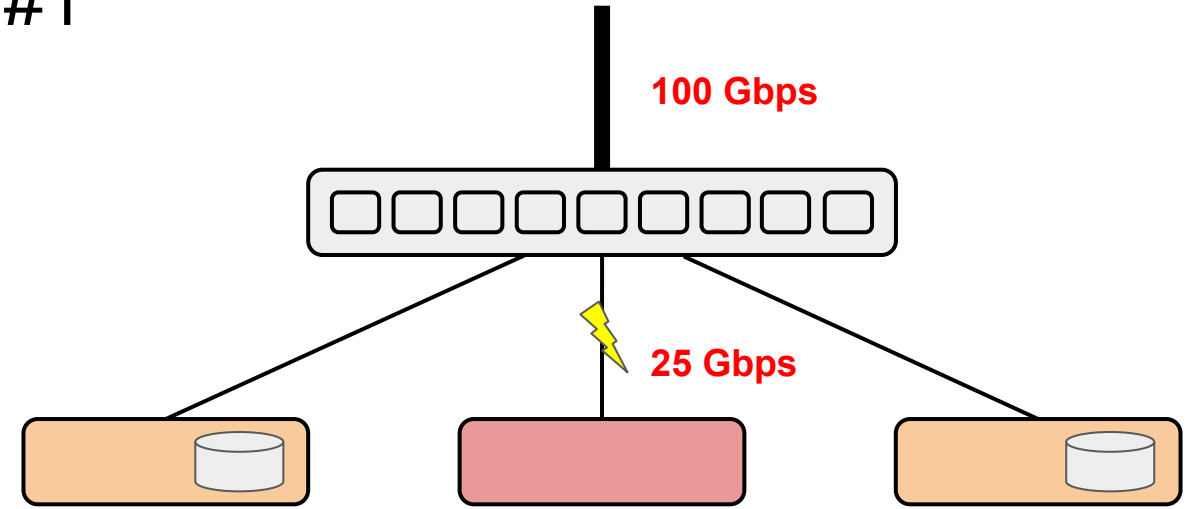

StorageGRID

SWIFT
an OpenStack Community Project

HAPROXY

Squid Proxy

# Problem #1



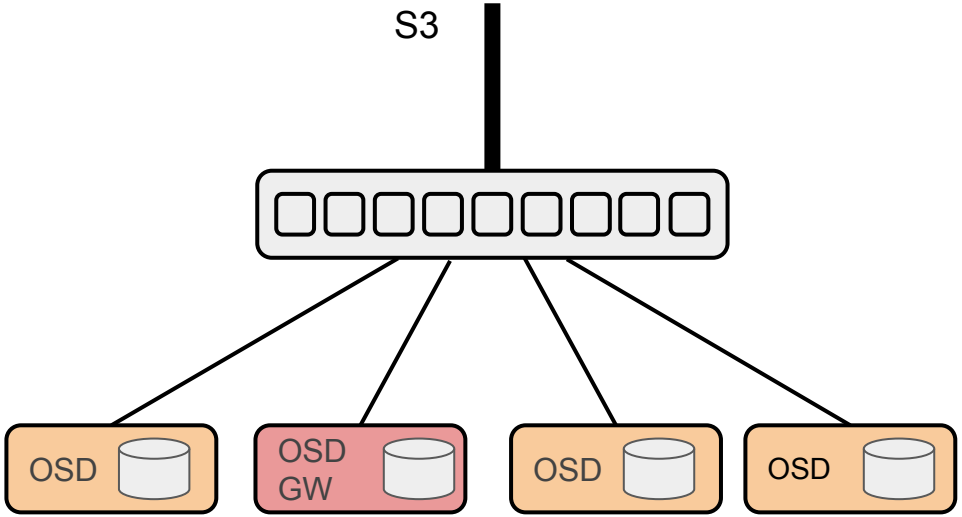

StorageGRID

SWIFT
an OpenStack Community Project

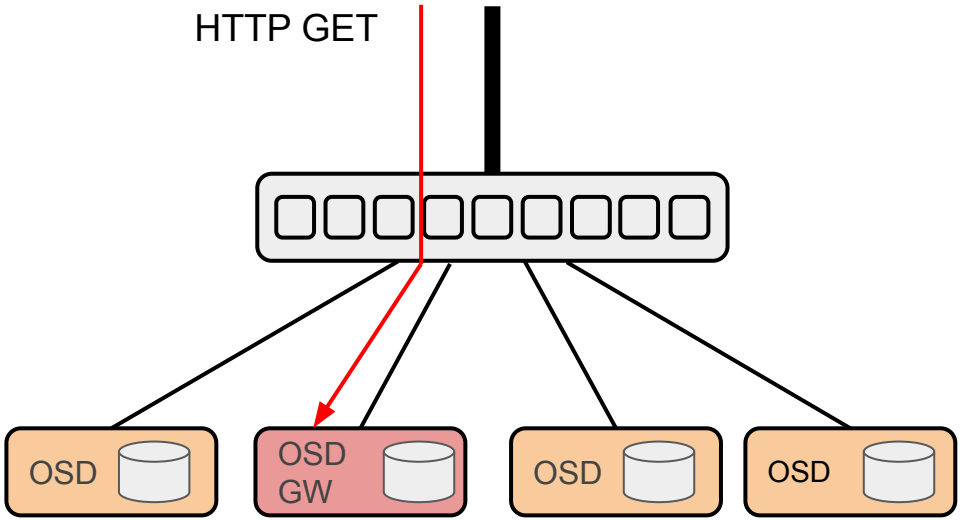
HAPROXY

Squid Proxy

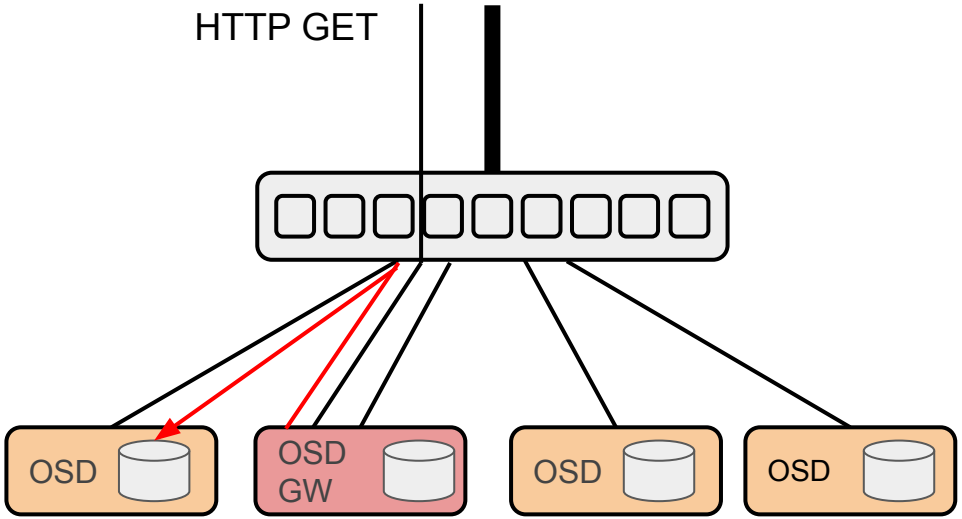
# Problem #2



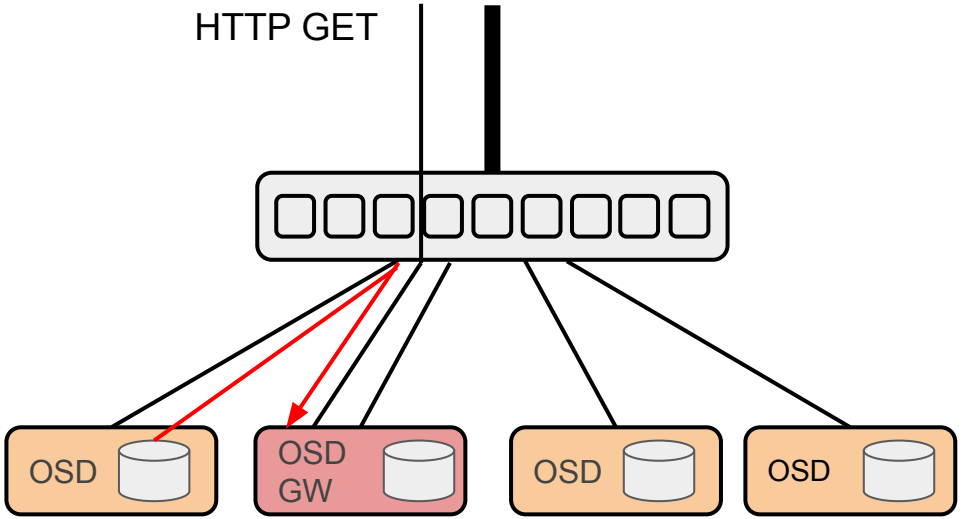
# Problem #2



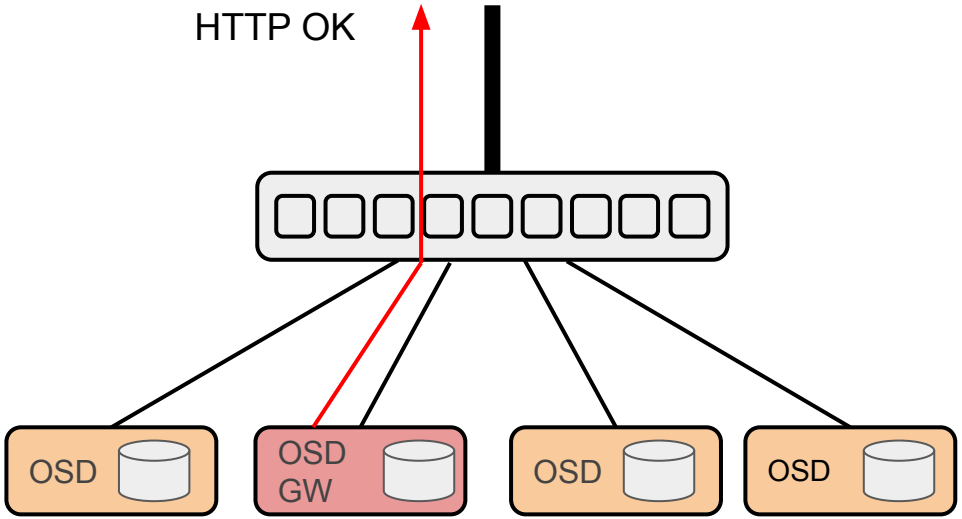
# Problem #2



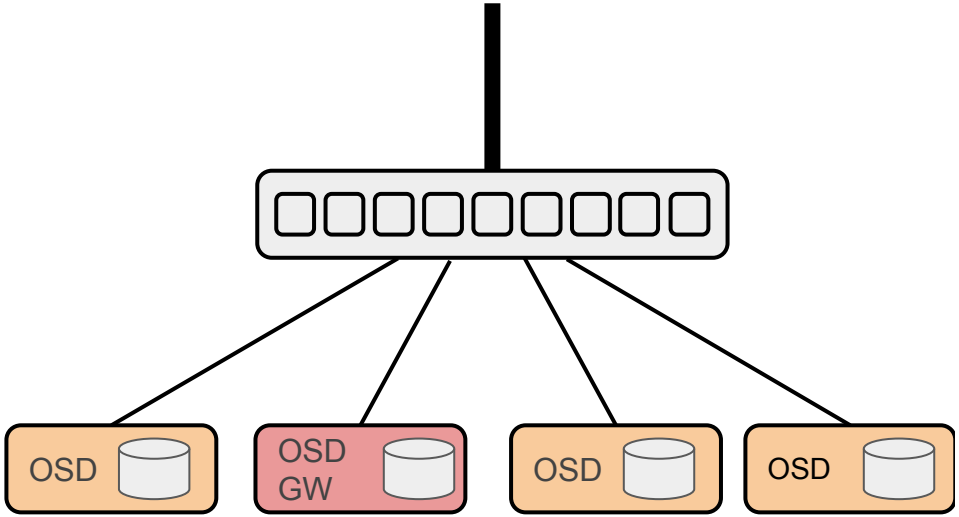
# Problem #2

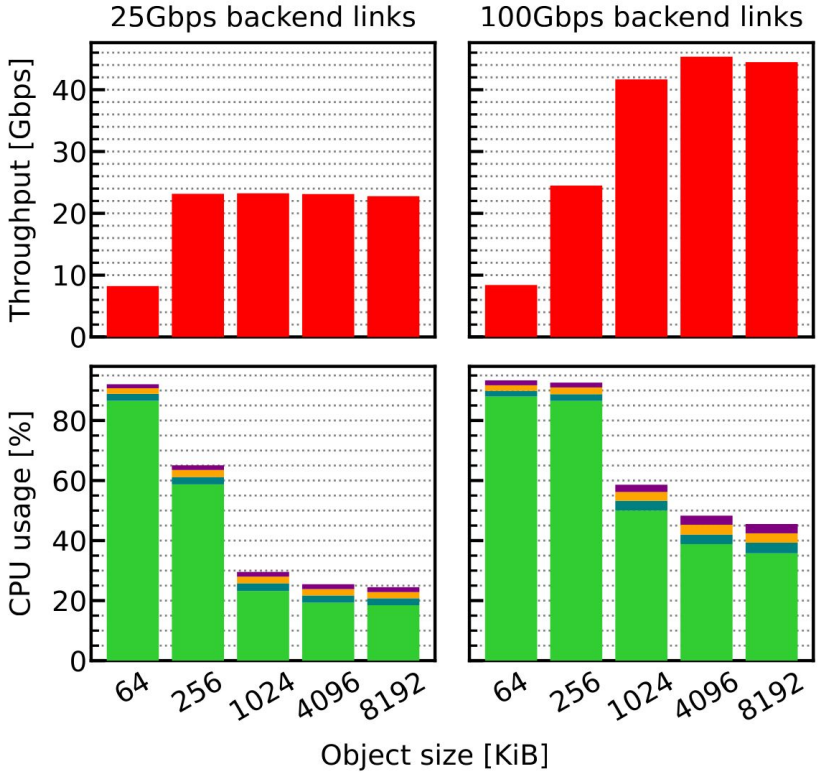


# Problem #2

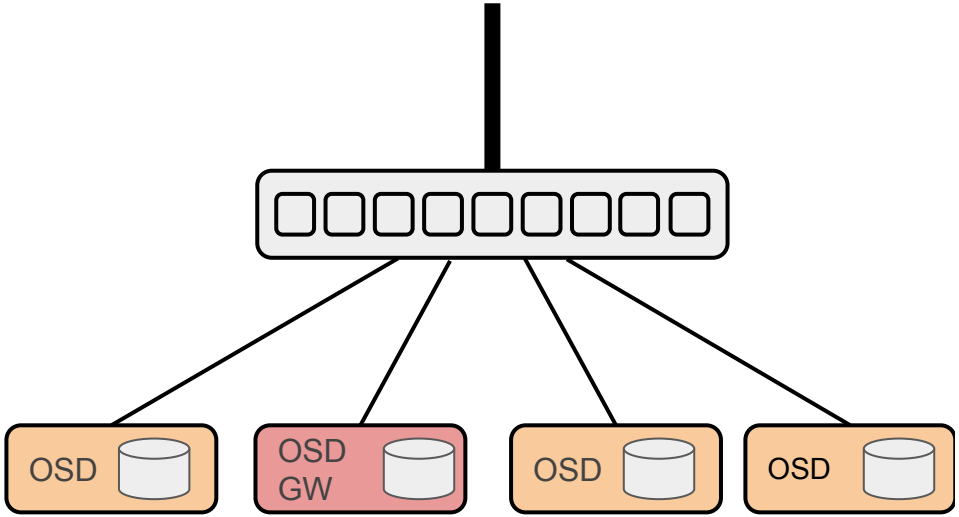


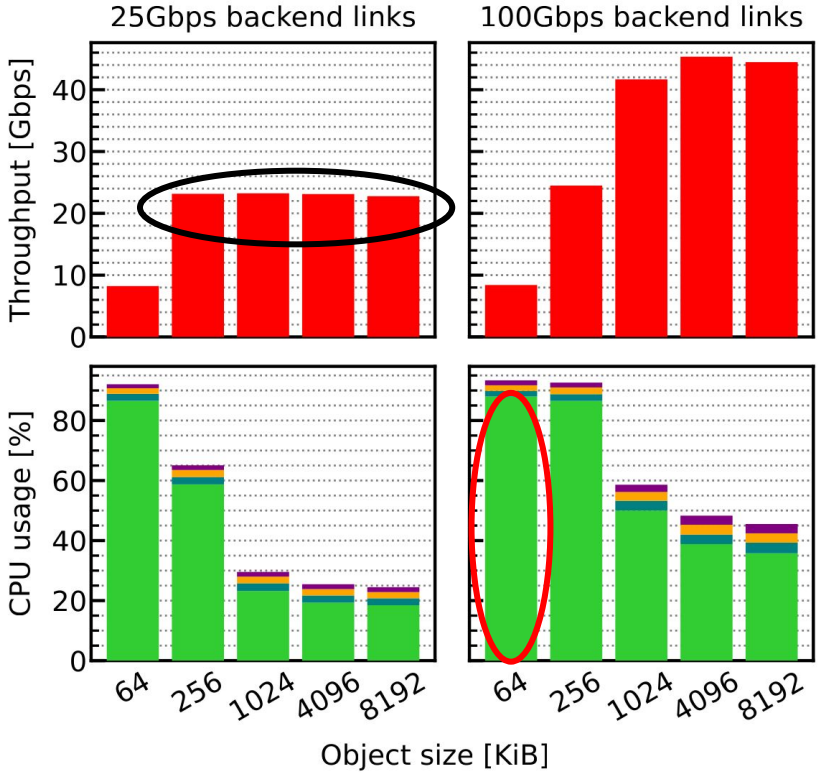
# Problem #2


OSD
OSD
GW
OSD
OSD


25Gbps backend links
100Gbps backend links
Throughput [Gbps]
0
10
20
30
40
CPU usage [%]
0
20
40
60
80
64
256
1024
4096
8192
Object size [KiB]

# Problem #2


OSD
OSD
GW
OSD
OSD


25Gbps backend links
100Gbps backend links
Throughput [Gbps]
0
10
20
30
40
CPU usage [%]
0
20
40
60
80
64
256
1024
4096
8192
Object size [KiB]

# Options…

- ~~L7 Load Balancer~~
  - Request oriented
  - L7 Proxy bandwidth - application level
- Splicing
  - Connection oriented
  - L7 Proxy bandwidth
  - Can’t touch the payload
  - Maybe require smart NIC for high performance

# Options…

- Content Aware Routing
  - Limited request response size
  - No encryption
  - Requires programmable switch
- Connection Migration
  - Request oriented
  - Line rate performance
  - Requires programmable switch

# XO - Crossover

# Requirements & Approaches

- No special hardware needed
  - Eliminate the need of programmable switch
- Transparency
  - Supports diverse packet filter methods
  - Reuse existing facilities e.g. eBPF+clsact , TC-Flower, …
- High-Performance
  - Unleash full bandwidth utilization through direct server response (DSR)

# TCP Connection State Offload

# Challenge #1

- Connection migration requires many non-atomic operations
  - **TCP/TLS connection serialization (many syscalls)**
  - NIC configuration (many syscalls and device configuration)
  - Inter-host signalling (many RPCs)

```
setsockopt(TCP_REPAIR)
ktls_serialize()
getsockopt(queue)
getsockopt(seq_no)
…
```

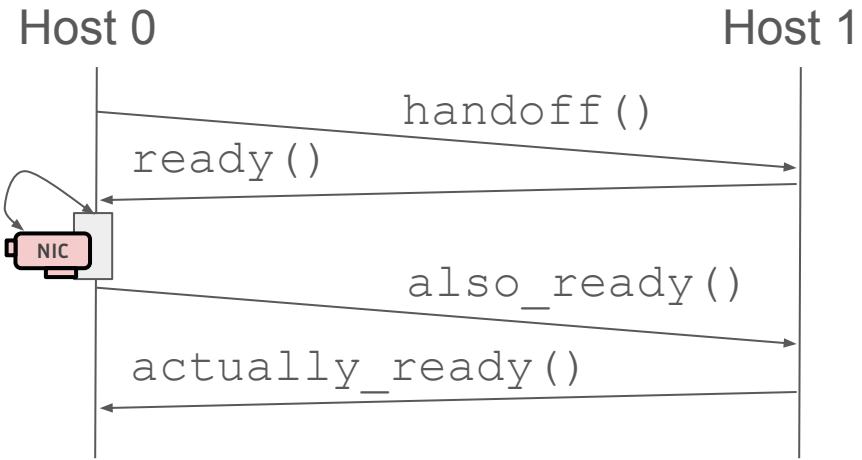
# Challenge #1

- Connection migration requires many non-atomic operations
  - TCP/TLS connection serialization (many syscalls)
  - **Forwarding and NIC configuration (many syscalls and device configuration)**
  - Inter-host signalling (many RPCs)

```
Netlink
    TCA_CLS_FLAGS_SKIP_SW/HW…
    TCA_FLOWER_…
    TCA_FLOWER_ACT
        …
```

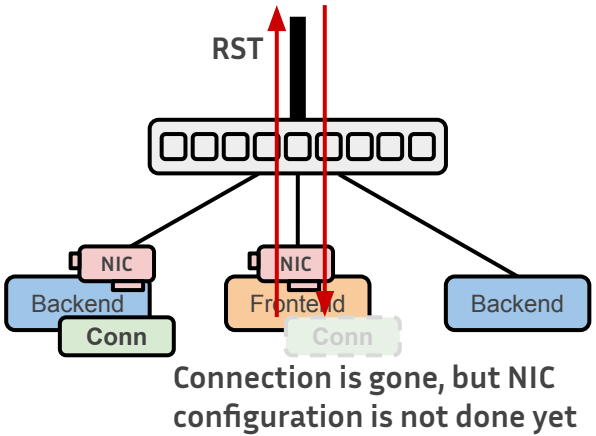
# Challenge #1

- Connection migration requires many non-atomic operations
  - TCP/TLS connection serialization (many syscalls)
  - NIC configuration (many syscalls and device configuration)
  - **Inter-host signalling (many RPCs)**

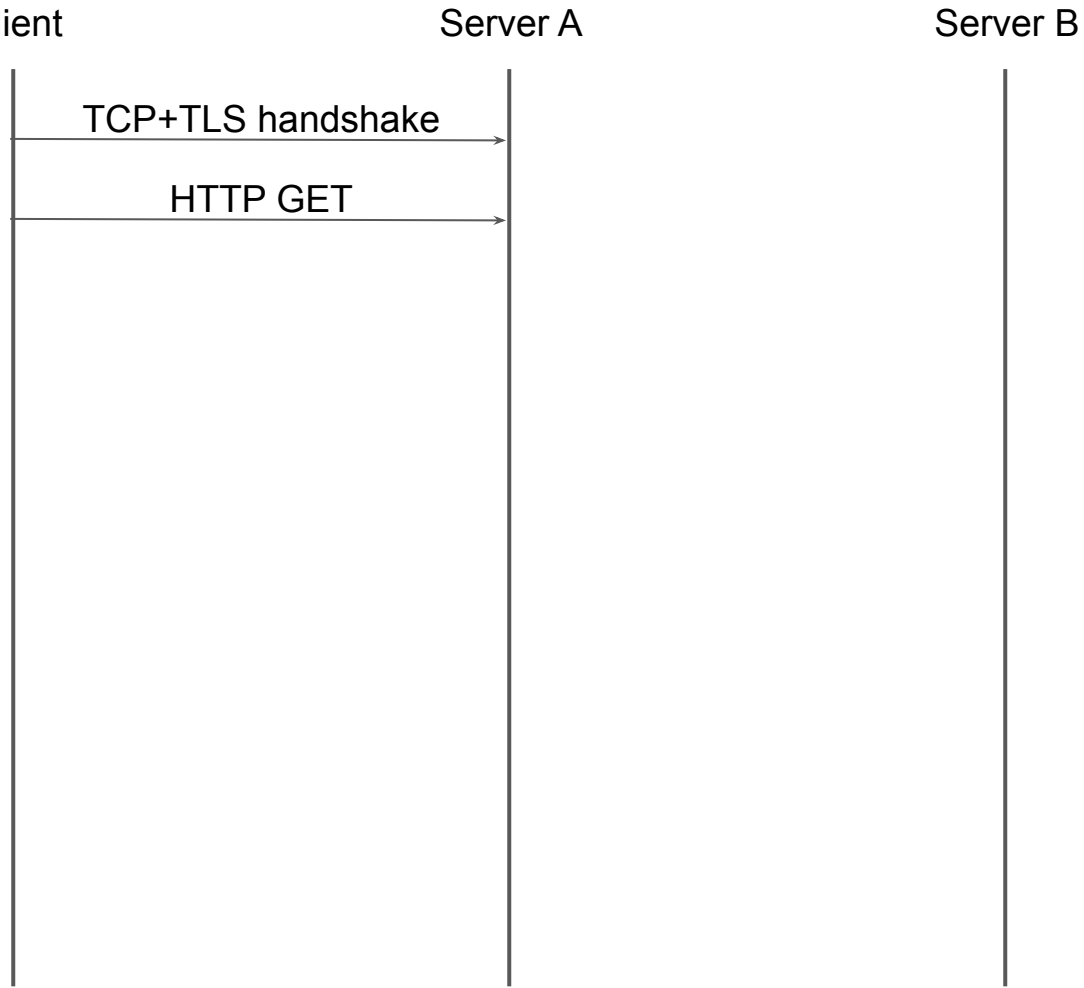


# Challenge #2

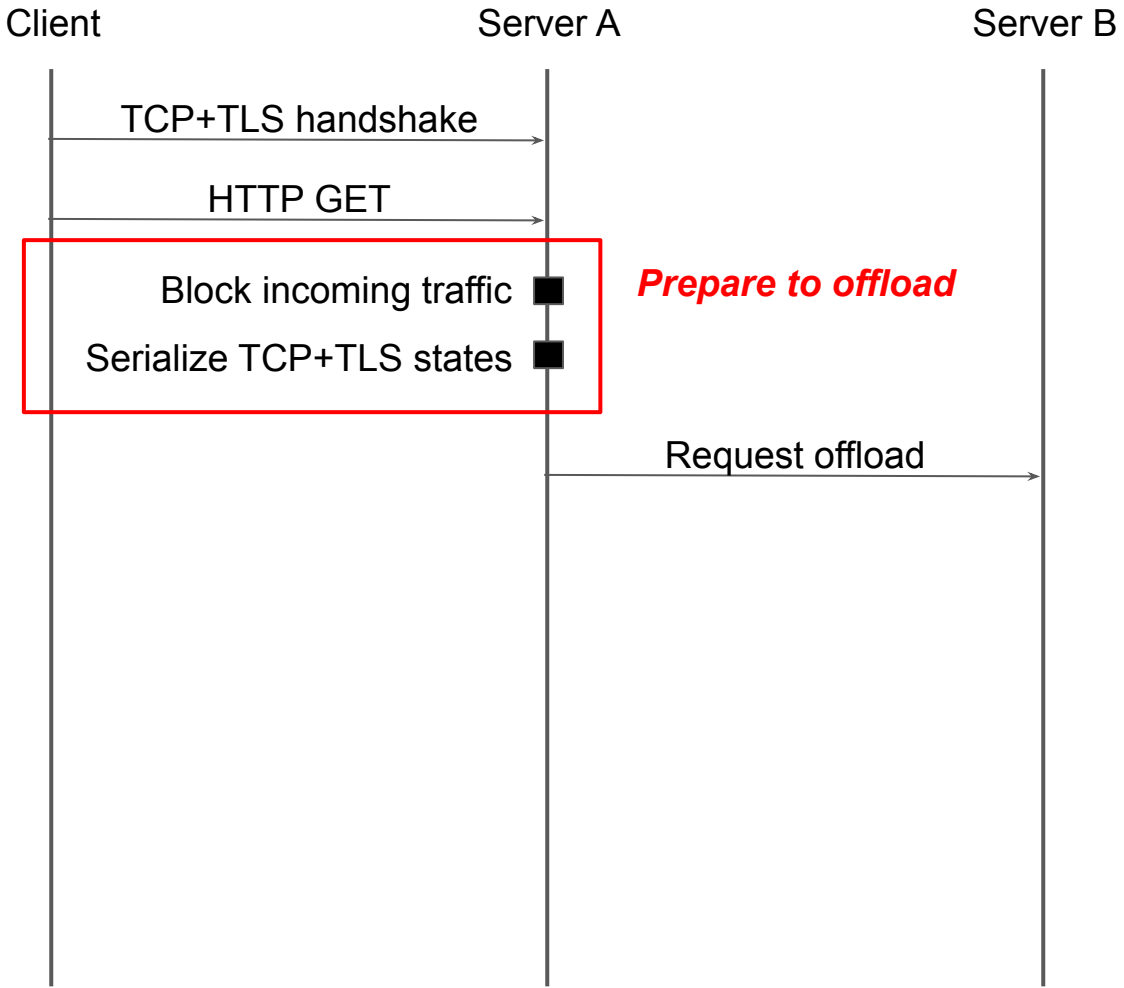
- Ingress and egress packets can break the connection
  - A socket is gone as soon as entering `TCP_REPAIR` mode
  - Treats incoming packets as error and issue `TCP_RST`



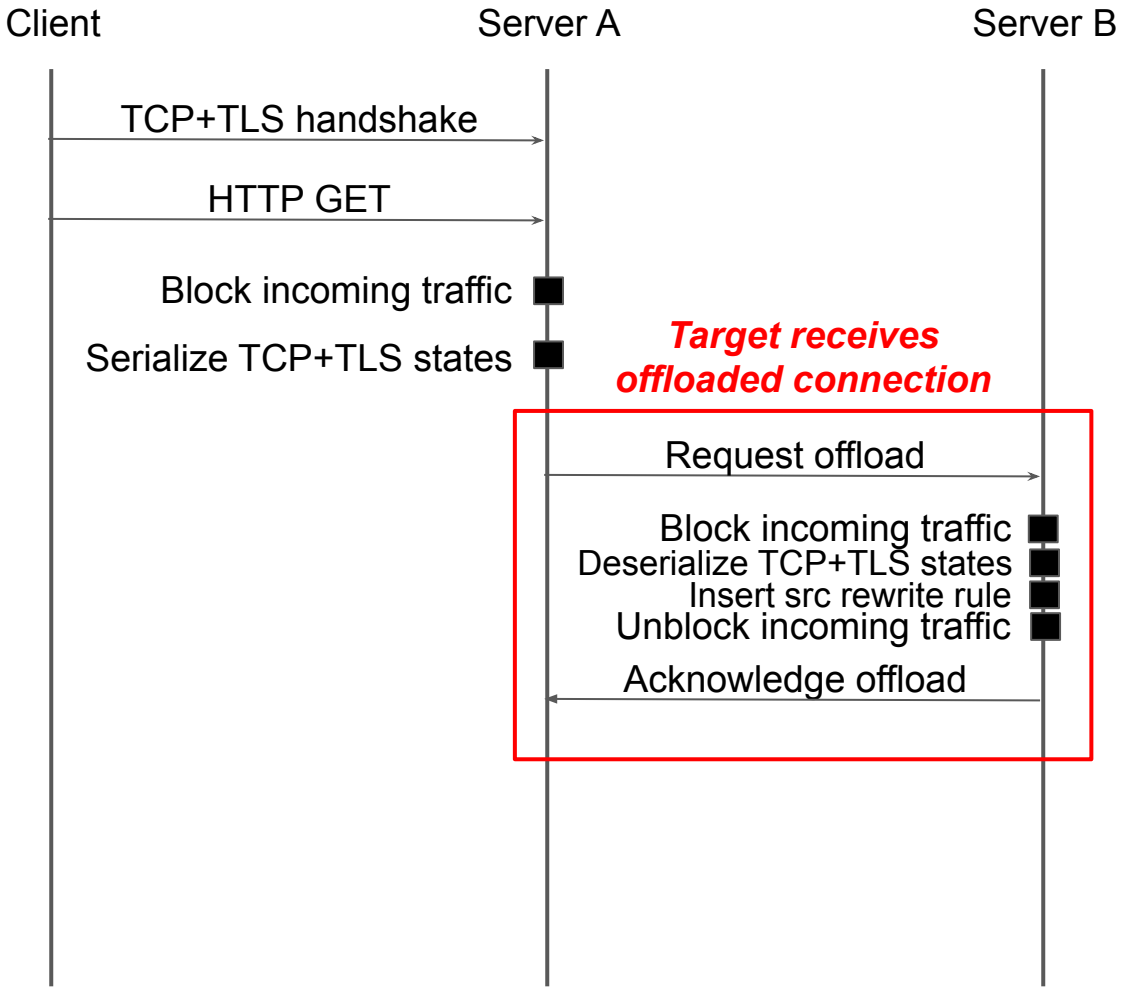
# Offload Protocol



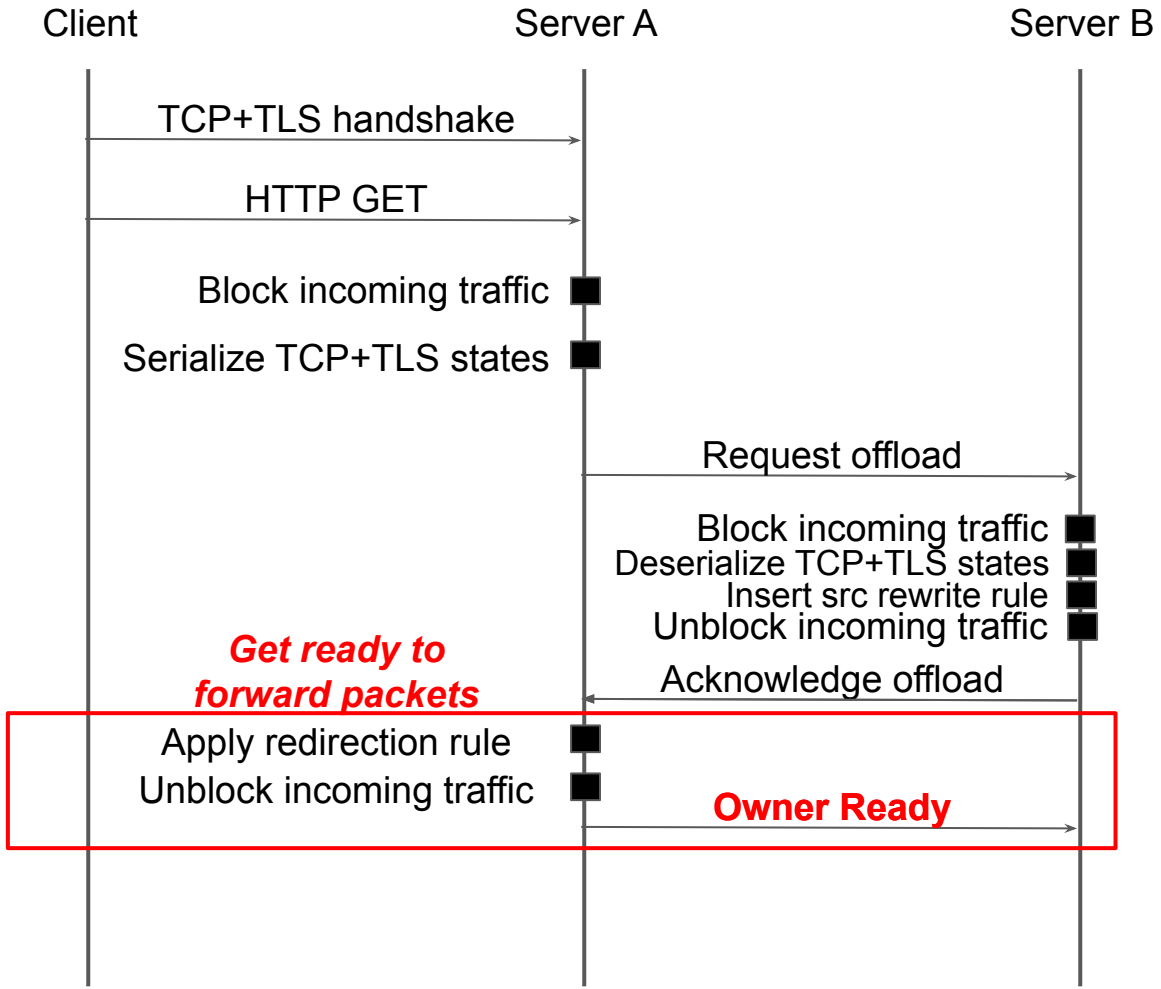
# Offload Protocol



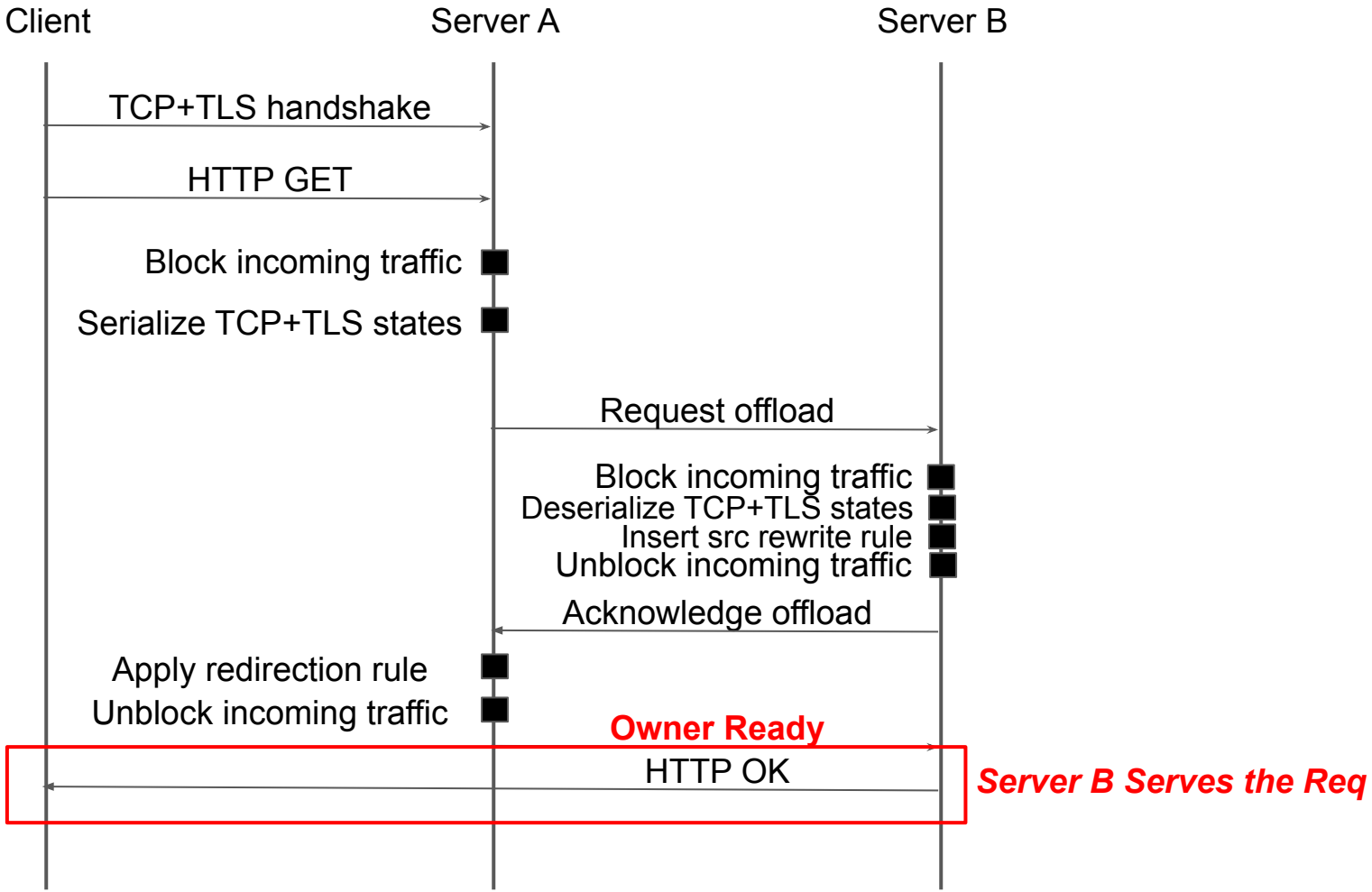
# Offload Protocol



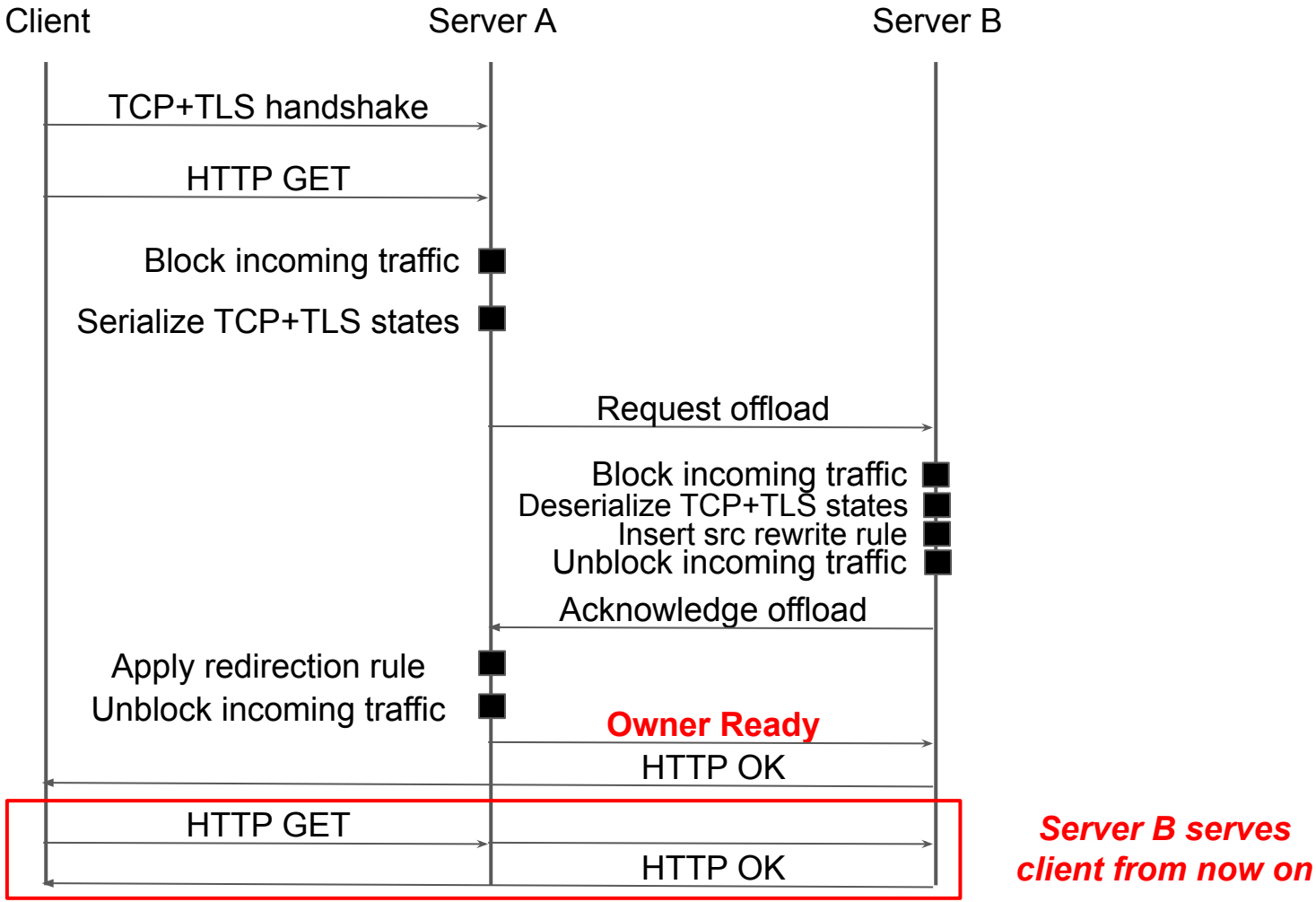
# Offload Protocol

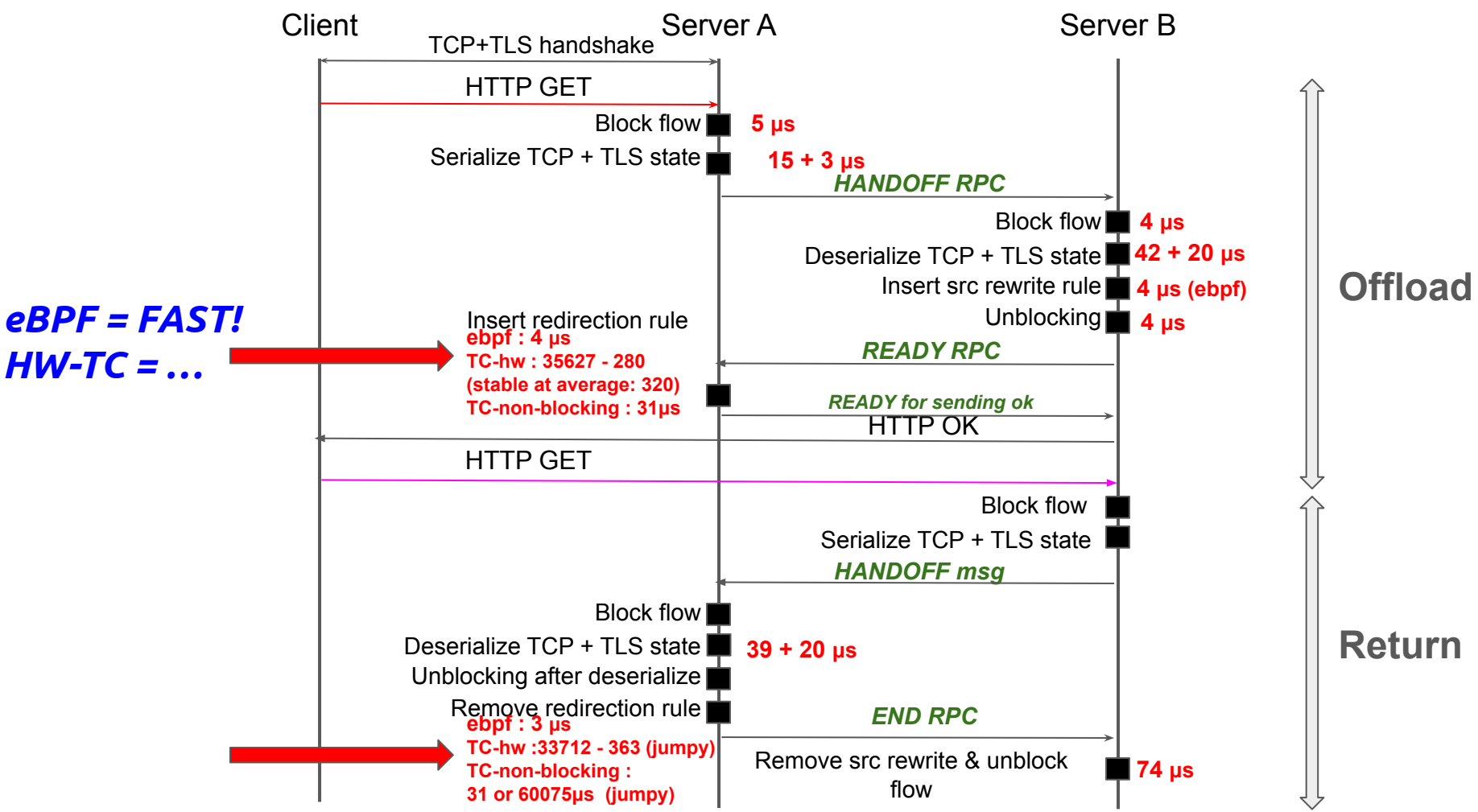


# Offload Protocol



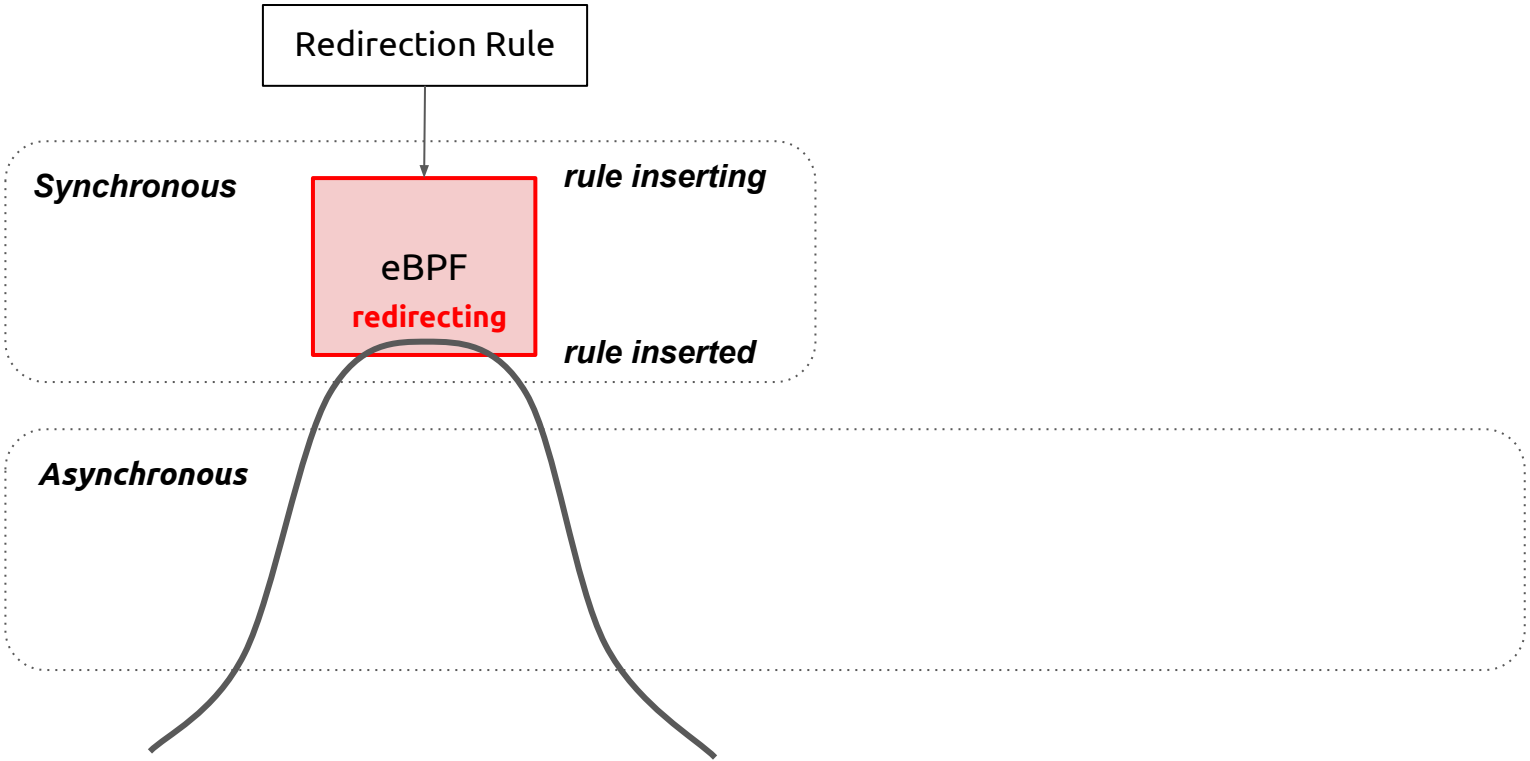
# Offload Protocol




Client
Server A
Server B
TCP+TLS handshake
HTTP GET
Block flow
5 µs
Serialize TCP + TLS state
15 + 3 µs
HANDOFF RPC
Block flow
4 µs
Deserialize TCP + TLS state
42 + 20 µs
Insert src rewrite rule
4 µs (ebpf)
Unblocking
4 µs
Offload
eBPF = FAST!
HW-TC = ...
Insert redirection rule
ebpf : 4 µs
TC-hw : 35627 - 280
(stable at average: 320)
TC-non-blocking : 31µs
READY RPC
READY for sending ok
HTTP OK
HTTP GET
Block flow
Serialize TCP + TLS state
HANDOFF msg
Block flow
Deserialize TCP + TLS state
39 + 20 µs
Unblocking after deserialize
Remove redirection rule
ebpf : 3 µs
TC-hw :33712 - 363 (jumpy)
TC-non-blocking :
31 or 60075µs (jumpy)
END RPC
Remove src rewrite & unblock flow
74 µs
Return

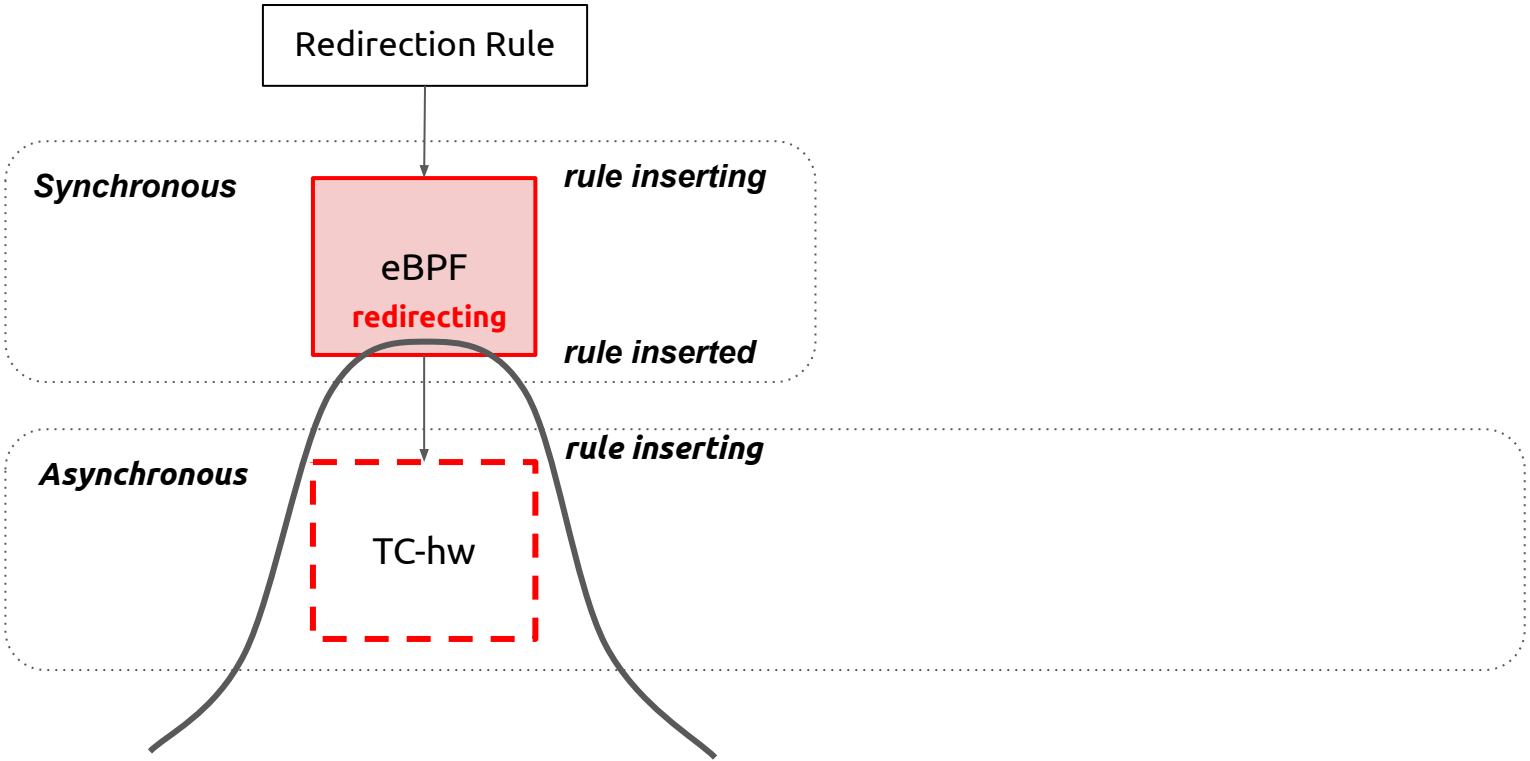
| | Operation (µs) | | Rate (Mpps) | | Latency (µs) | |
|---|---|---|---|---|---|---|
| :) | **Insert** | **Remove** | **64B** | **1500B** | **64B** | **1500B** |
| **eBPF (tc)** | **4.01** | **3.77** | **0.79** | **0.78** | 21.06 | 22.42 |
| **eBPF (XDP)** | 38.31 | 7.41 | 6.65 | 2.07 | 16.52 | 18.45 |
| :( **TC (CX5)** | **476** | **404** | **33.01** | **2.07** | 8.26 | 9.89 |
| **TC (CX7)** | **2143** | **1134** | **33.08** | **2.07** | 8.41 | 9.97 |
| **TC (Agilio)** | 68 | 65 | 22.12 | 2.07 | 19.77 | 20.58 |

# HW-SW Hybrid Packet Redirection



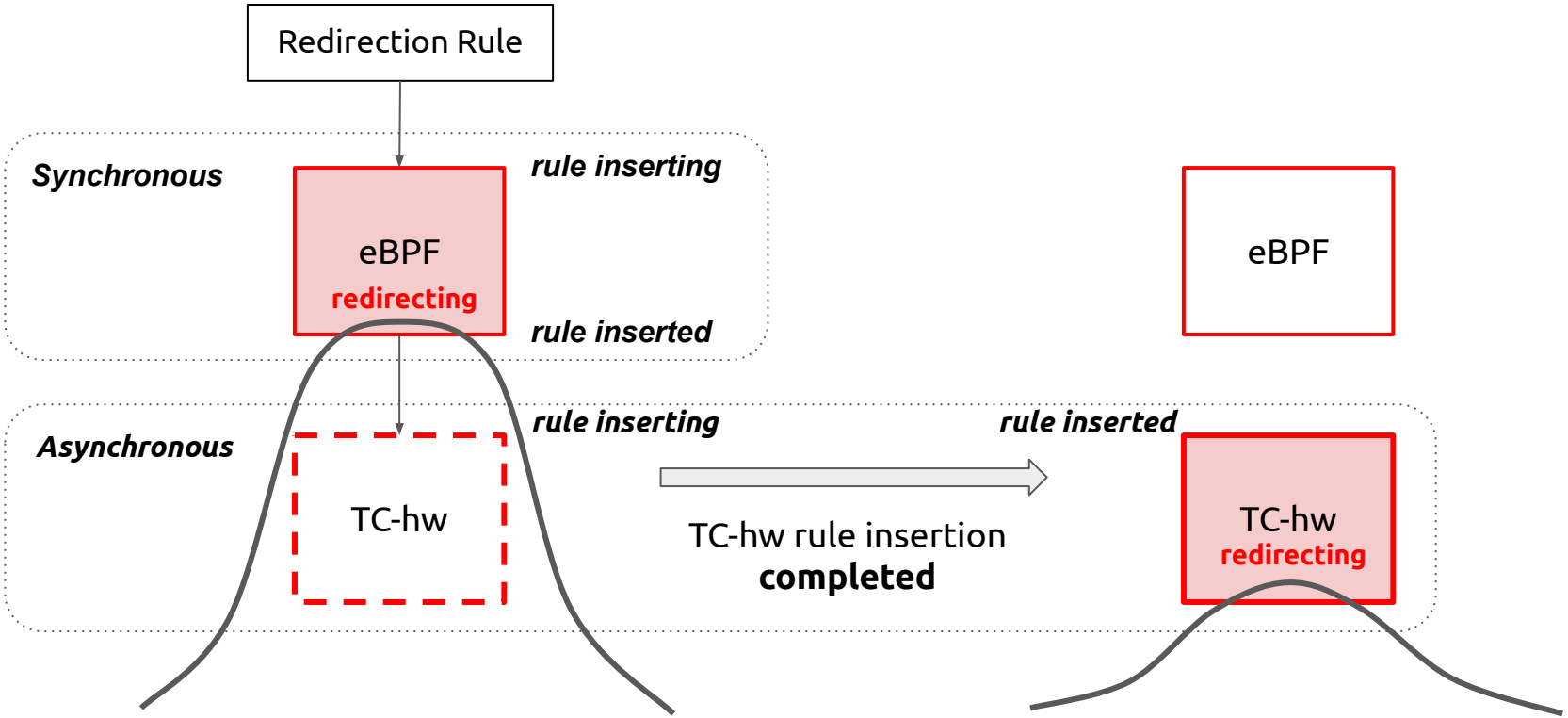
# HW-SW Hybrid Packet Redirection

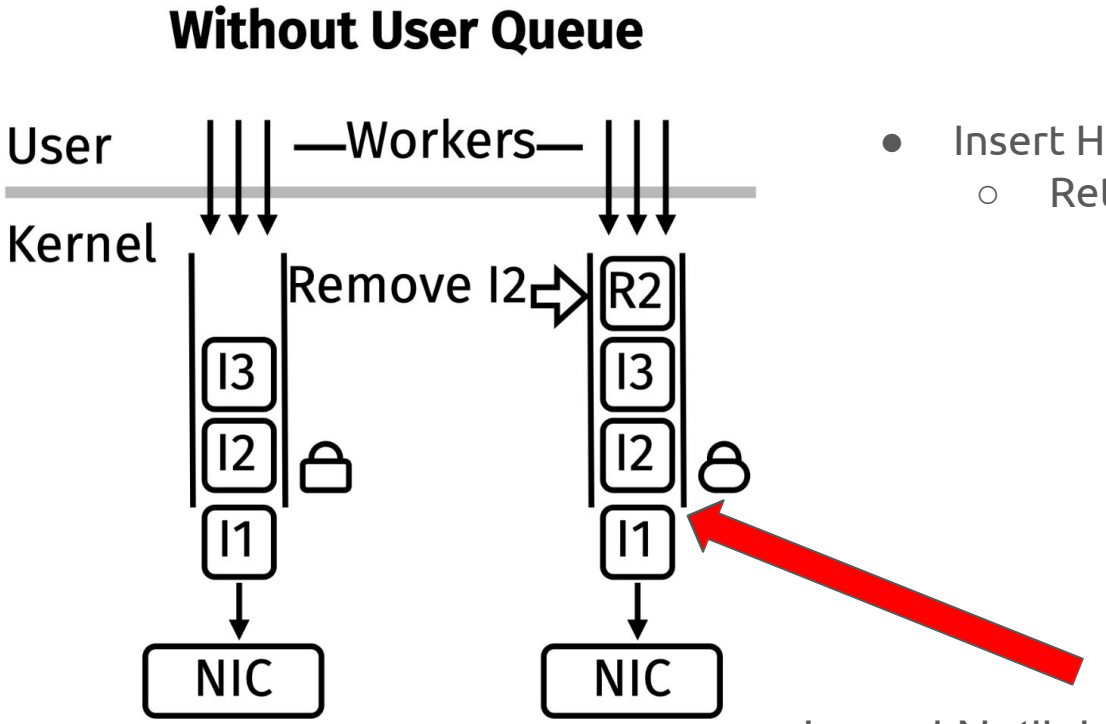
**Use eBPF-based redirection until the HW one is activated**



# HW-SW Hybrid Packet Redirection

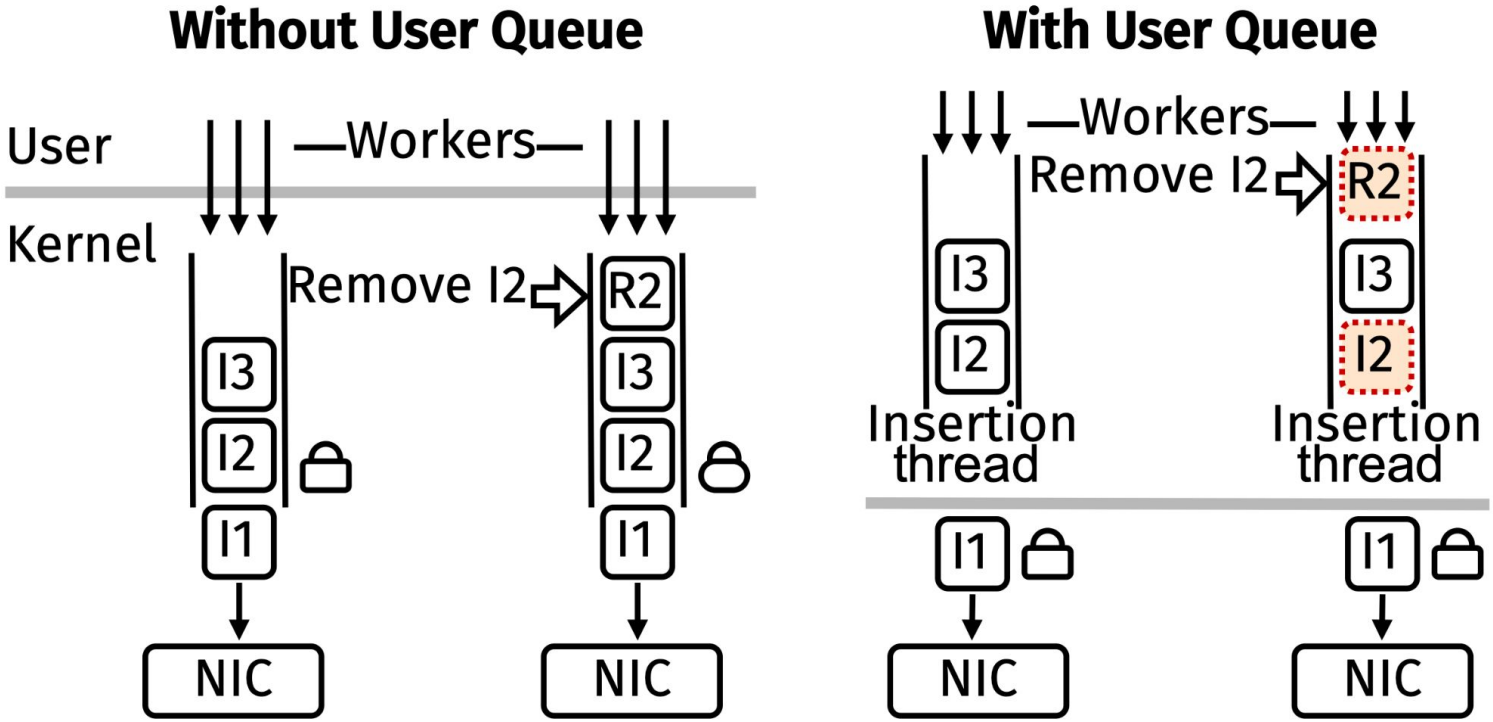
**Use eBPF-based redirection until the HW one is activated**






- Insert HW rules with background thread
  - Returns immediately after eBPF

Issued Netlink request cannot be cancelled!!


Without User Queue
User
Kernel
—Workers—
Remove I2
R2
I3
I2
I1
NIC
NIC
With User Queue
—Workers—
Remove I2
R2
I3
I2
Insertion thread
Insertion thread
I1
I1
NIC
NIC

# One more small problem …

```
int sendq_len, unsentq_len, recvq_len;
ret = ioctl(fd, SIOCOUTQ, &sendq_len);
assert(ret == 0);
```

**Check if TCP queues are empty…**

……

```
const int peek = MSG_PEEK | MSG_DONTWAIT;
uint8_t *sndbuf = NULL;
if (sendq_len)
{
        sndbuf = calloc(1, sendq_len + 1);
        assert(sndbuf != NULL);
        ret = recv(fd, sndbuf, sendq_len + 1, peek);
        assert(ret == sendq_len);
}
```

# One more small problem …

```
int sendq_len, unsentq_len, recvq_len;
ret = ioctl(fd, SIOCOUTQ, &sendq_len);
assert(ret == 0);

 ……

const int peek = MSG_PEEK | MSG_DONTWAIT;
uint8_t *sndbuf = NULL;
if (sendq_len)
{
        sndbuf = calloc(1, sendq_len + 1);
        assert(sndbuf != NULL);
        ret = recv(fd, sndbuf, sendq_len + 1, peek);
        assert(ret == sendq_len);
}
```

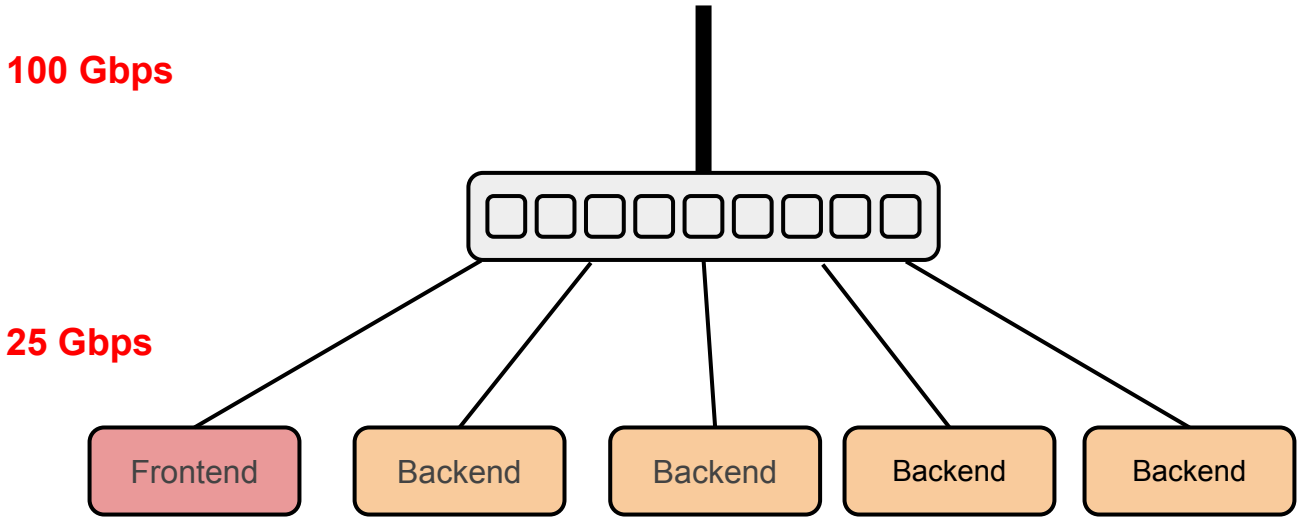
**Preserve them…**

# One more small problem …

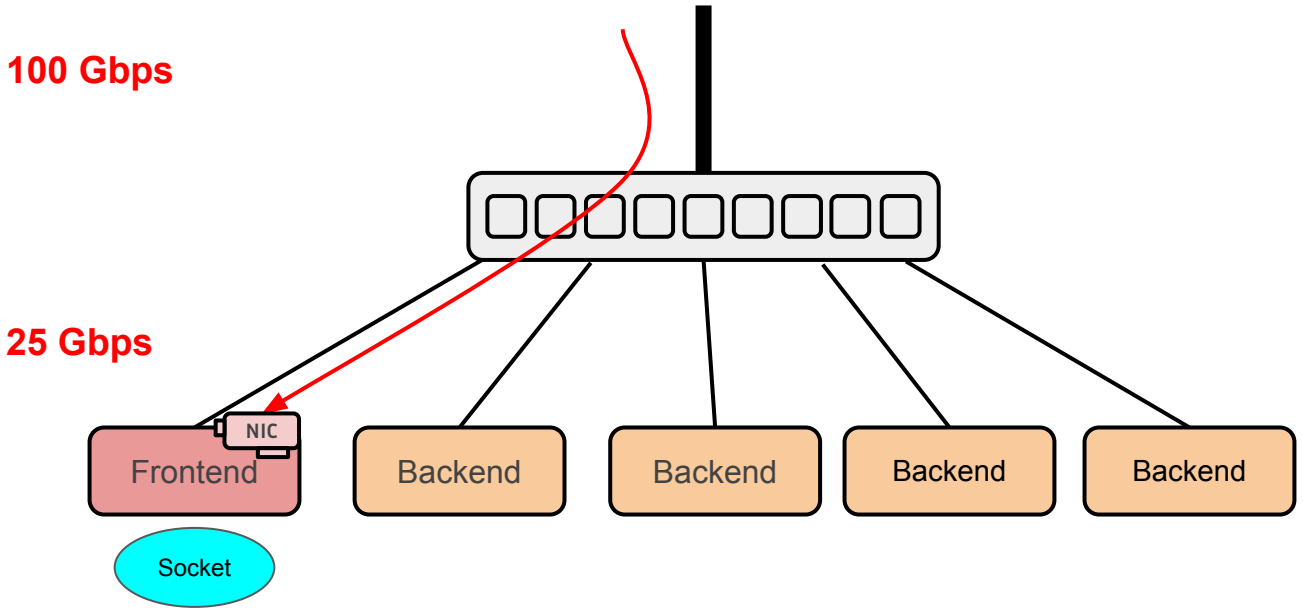
```
TCP REPAIR needs to read the data in the send or receive queue.
This patch forwards those to TCP.

Signed-off-by: Michio Honda <michio.honda@ed.ac.uk>
---
 net/tls/tls_sw.c | 8 ++++++++
 1 file changed, 8 insertions(+)

diff --git a/net/tls/tls_sw.c b/net/tls/tls_sw.c
index 305a412785f5..25b239a9b748 100644
--- a/net/tls/tls_sw.c
+++ b/net/tls/tls_sw.c
@@ -1973,6 +1973,14 @@ int tls_sw_recvmsg(struct sock *sk,
        bool bpf_strp_enabled;
        bool zc_capable;

+       struct tcp_sock *tp = tcp_sk(sk);
+
+       if (unlikely(tp->repair)) {
+               if (tp->repair_queue == TCP_SEND_QUEUE ||
+                   tp->repair_queue == TCP_RECV_QUEUE)
+                       return tcp_recvmsg(sk, msg, len, flags, addr_len);
+       }
+
        if (unlikely(flags & MSG_ERRQUEUE))
                return sock_recv_errqueue(sk, msg, len, SOL_IP, IP_RECVERR);
```

**Need kTLS support for `TCP_REPAIR` socket…**

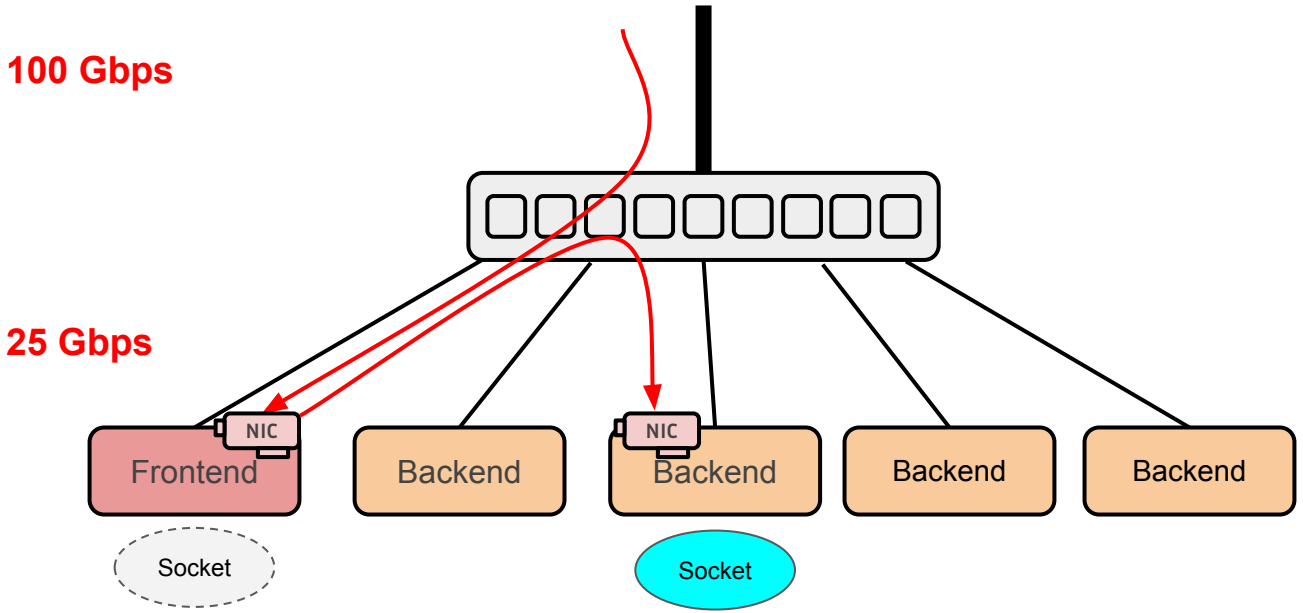
# Evaluation



# Evaluation

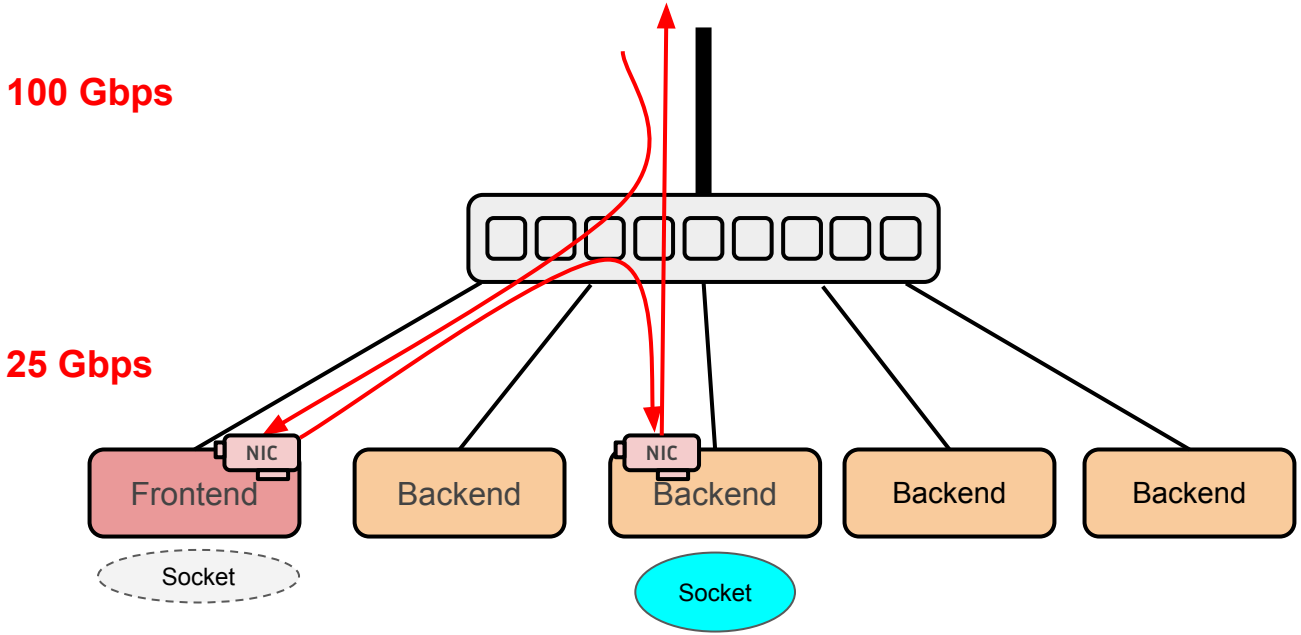

100 Gbps
25 Gbps
NIC
Frontend
Backend
Backend
Backend
Backend
Socket

# Evaluation

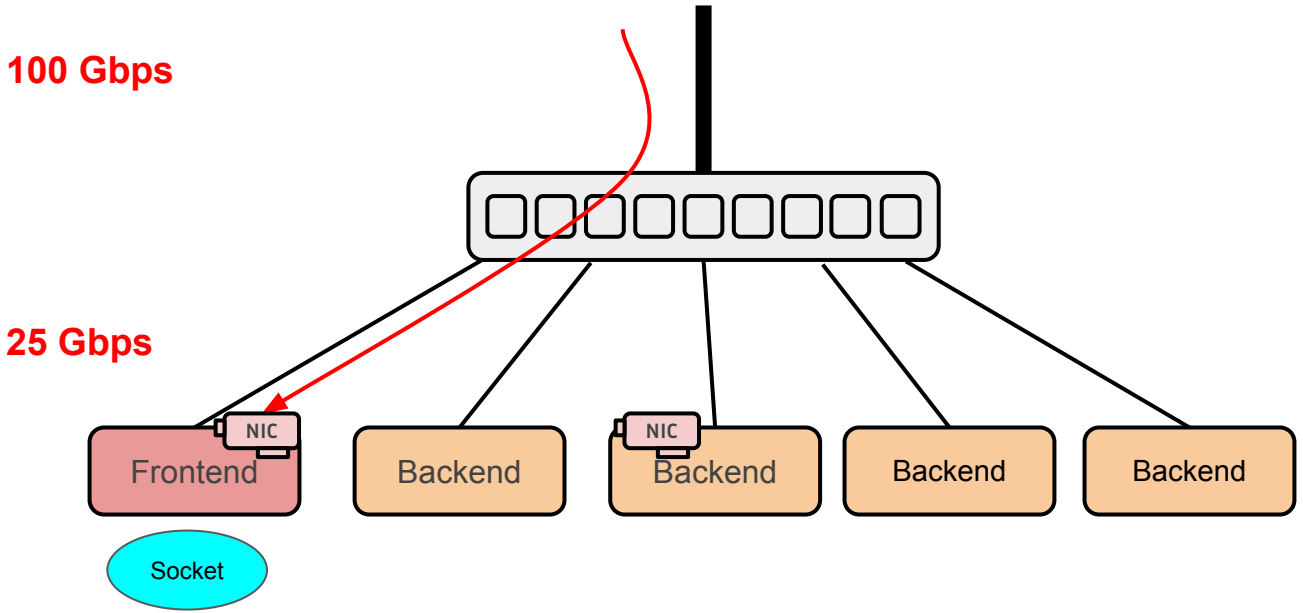



**Offload TCP connection…**

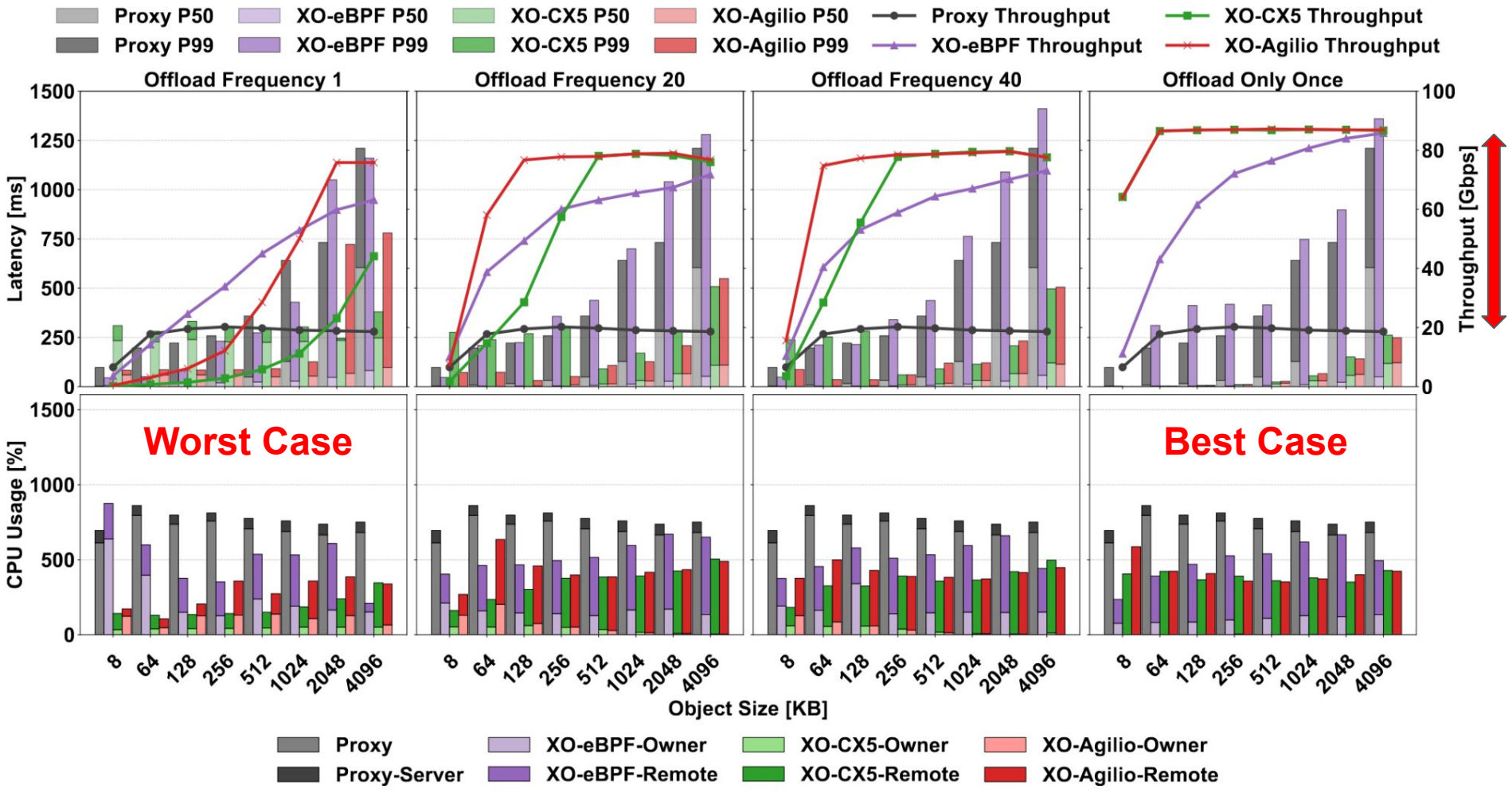
# Evaluation


100 Gbps
25 Gbps
NIC
NIC
Frontend
Backend
Backend
Backend
Backend
Socket
Socket

# Evaluation




**Terminate offload…**


Proxy P50
Proxy P99
XO-eBPF P50
XO-eBPF P99
XO-CX5 P50
XO-CX5 P99
XO-Agilio P50
XO-Agilio P99
Proxy Throughput
XO-eBPF Throughput
XO-CX5 Throughput
XO-Agilio Throughput
Offload Frequency 1
Offload Frequency 20
Offload Frequency 40
Offload Only Once
Latency [ms]
Throughput [Gbps]
CPU Usage [%]
Worst Case
Best Case
8
64
128
256
512
1024
2048
4096
Object Size [KB]
Proxy
Proxy-Server
XO-eBPF-Owner
XO-eBPF-Remote
XO-CX5-Owner
XO-CX5-Remote
XO-Agilio-Owner
XO-Agilio-Remote

# NGINX

# NGINX Integration

- Implementing NGINX as an HTTP module
  - Reuse event loop
  - Reuse HTTP module pipeline
  - Flexible offload and return policies
  - Minimal modification to core NGINX to track connection data

# NGINX Integration

```
handoff_ifname enp8s0f0np0;
handoff_freq 1000; # 0 = round robin
handoff_target 192.168.11.11 79;
handoff_target 192.168.11.33 79;
handoff_target 192.168.11.31 79;
handoff_target 192.168.11.53 79;
```

**Specify backends**

# NGINX Integration

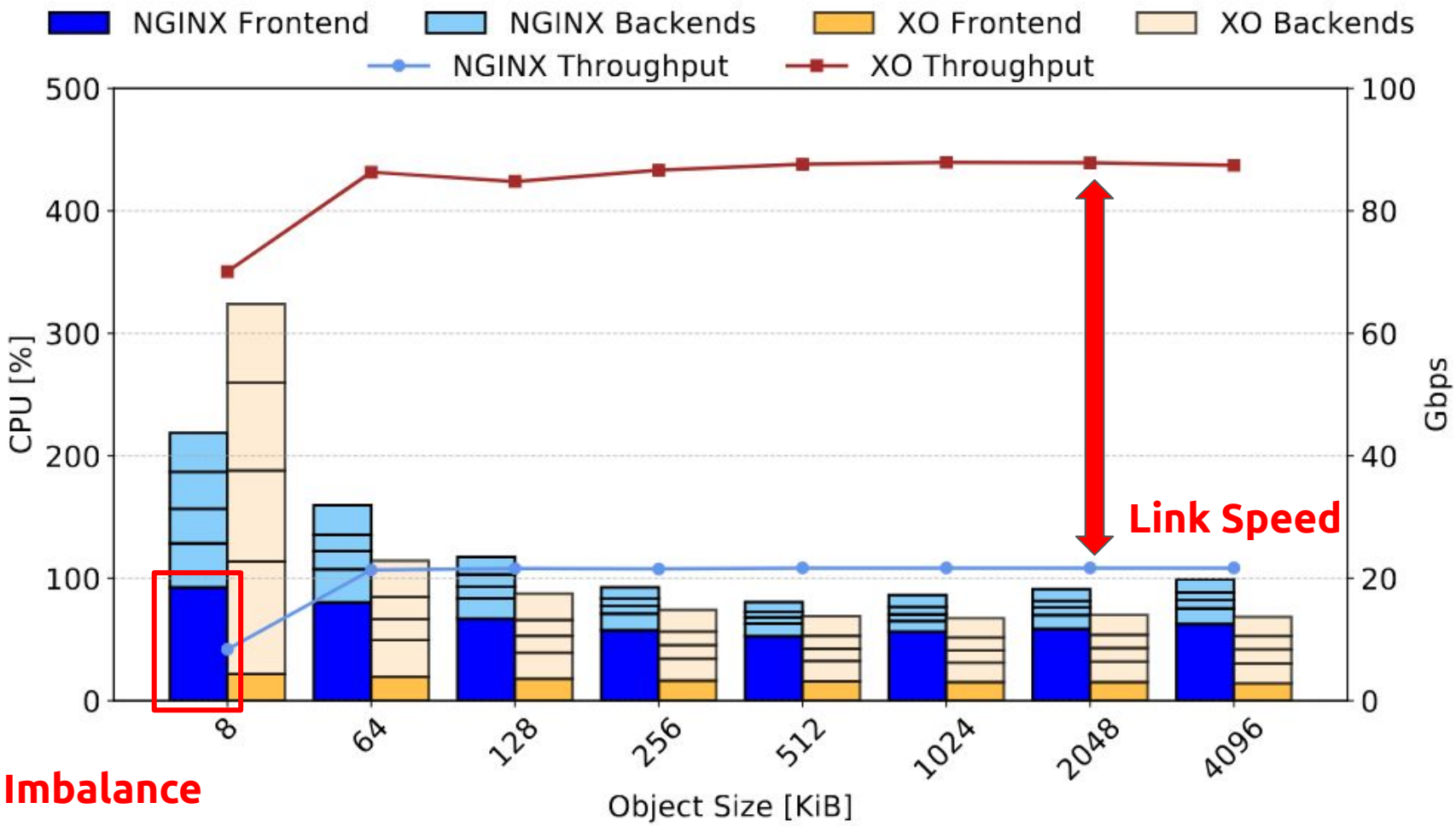
```
server {
    listen 80;
    location  / {
        handoff_out;
    }
}

server {
    listen 79;
    location  / {
        handoff_in;
    }
}
```

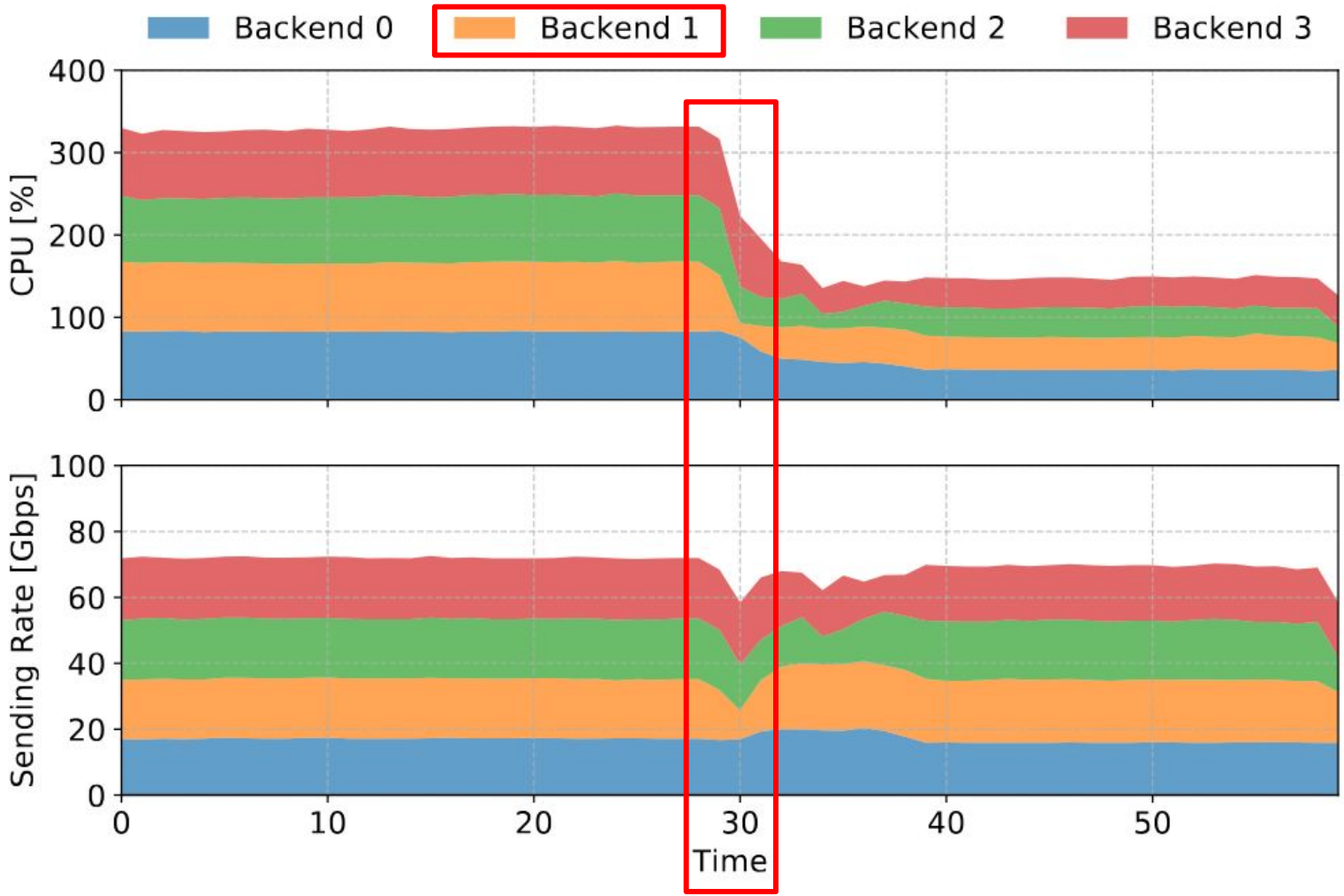
**Frontend**

**Backend / Control**

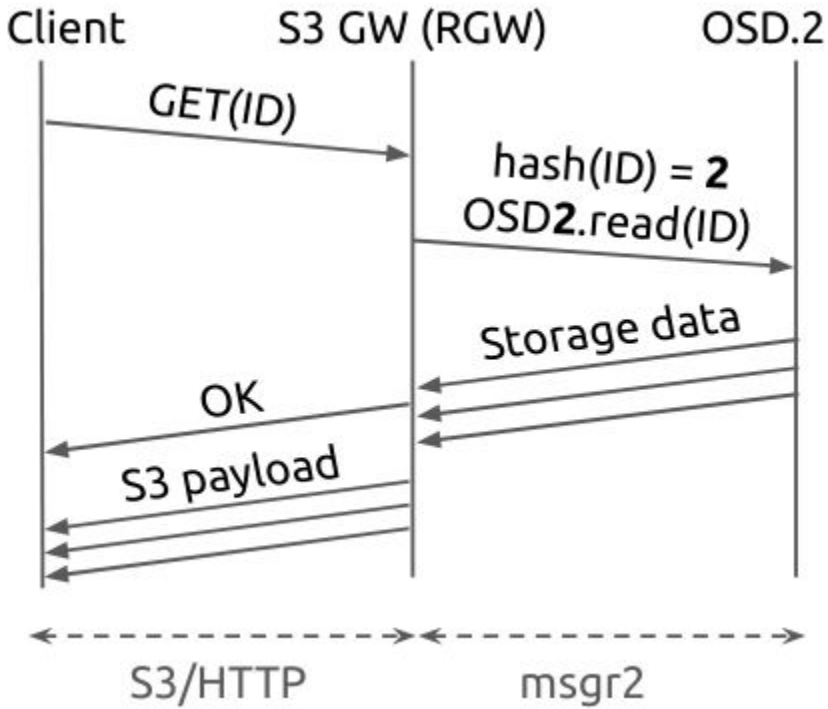
# NGINX Integration

1. Receiving incoming TCP conn via `handoff_out` port, and HTTP request
2. Serialize socket, open connection to a backend via `handoff_in` port
3. Execute offload protocol
4. Backend restores the socket, and mimics an accept
5. Future HTTP requests all handled by normal HTTP module code path

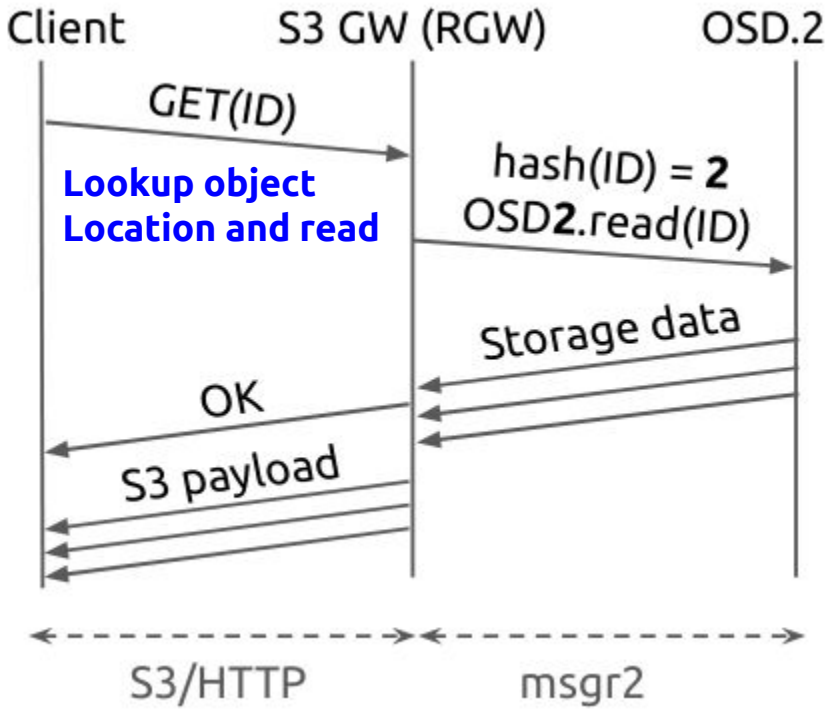

NGINX Frontend
NGINX Backends
XO Frontend
XO Backends
NGINX Throughput
XO Throughput
CPU [%]
Gbps
Object Size [KiB]
8
64
128
256
512
1024
2048
4096
0
100
200
300
400
500
0
20
40
60
80
100
Link Speed
Imbalance


Backend 0
Backend 1
Backend 2
Backend 3
CPU [%]
400
300
200
100
0
Sending Rate [Gbps]
100
80
60
40
20
0
0
10
20
30
40
50
Time

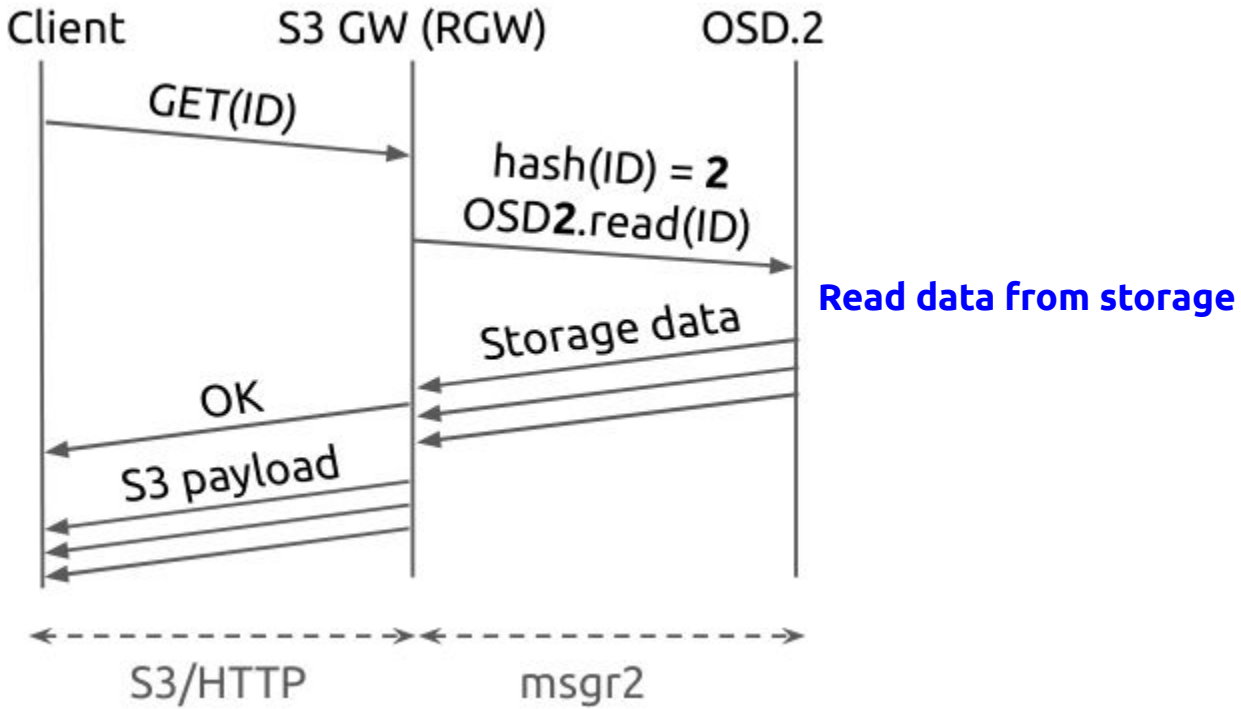
# Ceph Integration



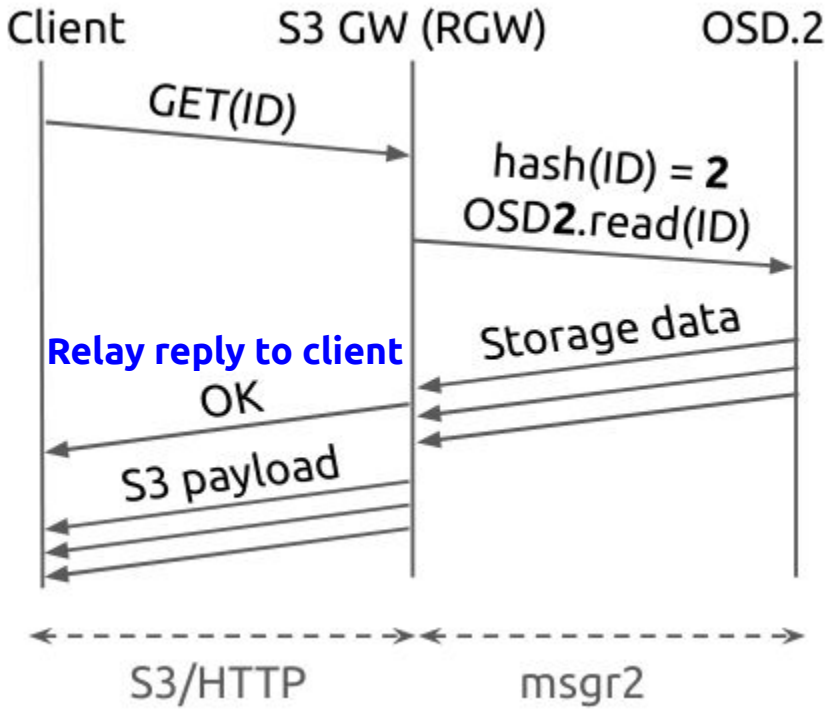
# Ceph Integration



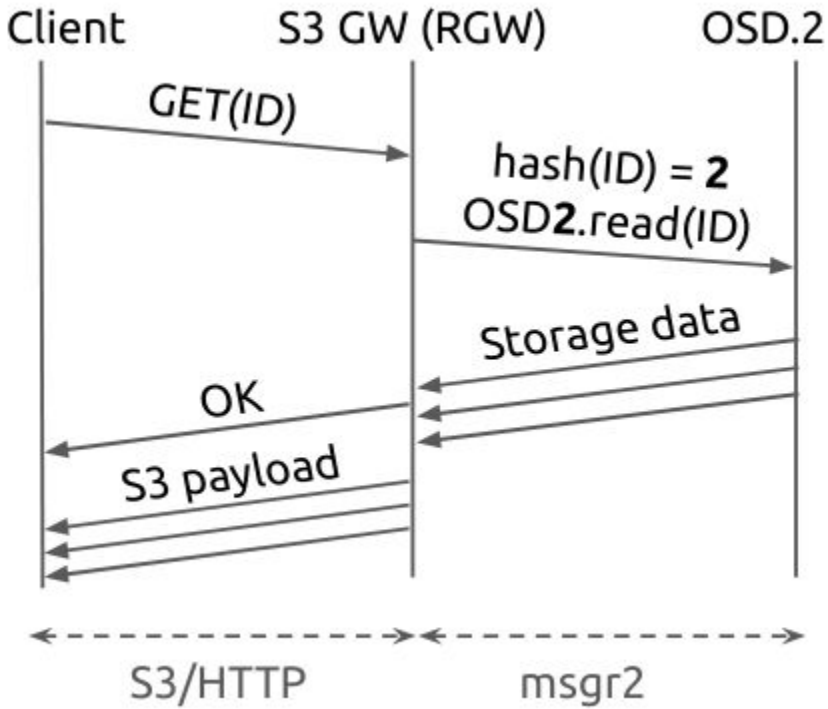
# Ceph Integration



# Ceph Integration



# Ceph Integration

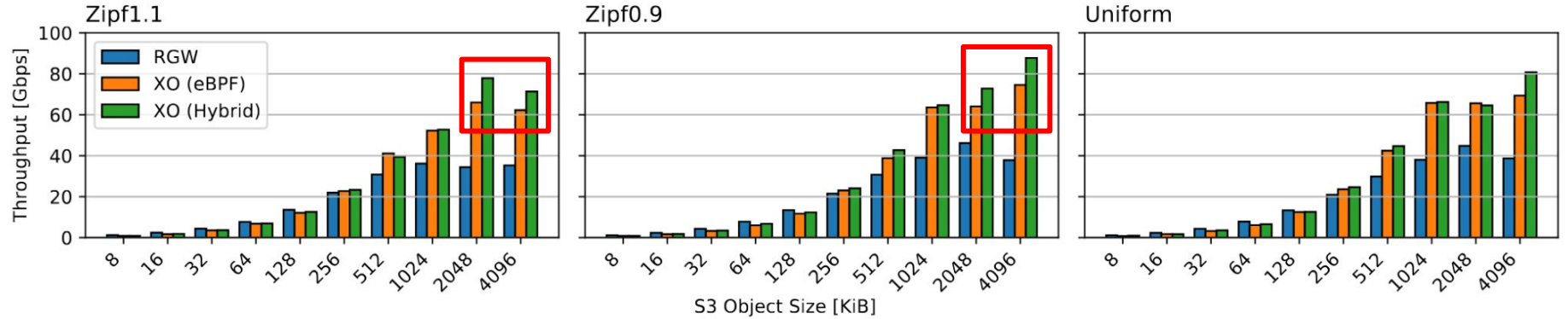


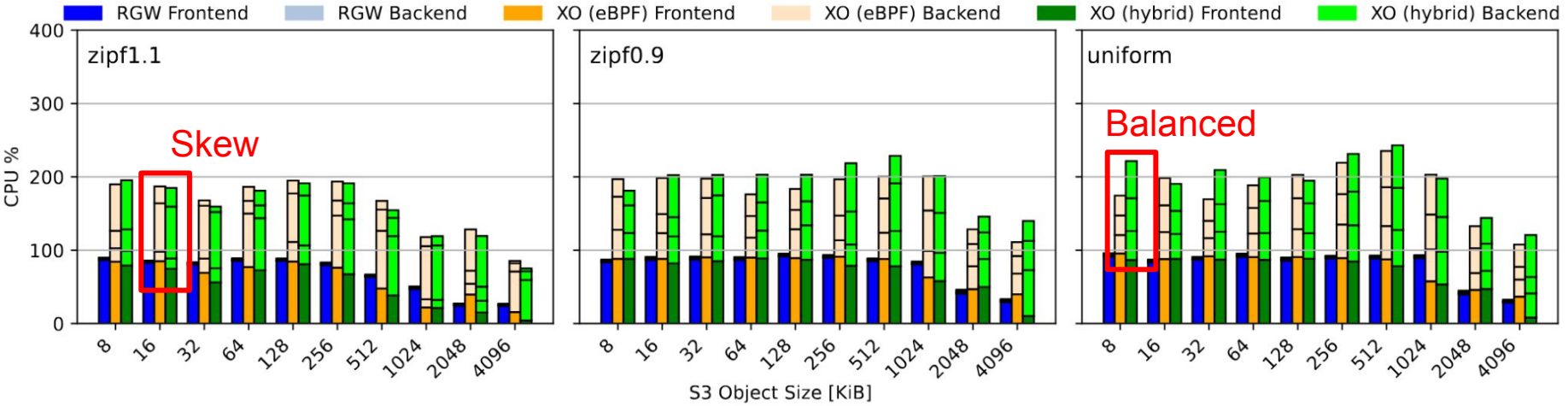

**Protocol translate: no splicing, no end-to-end encryption!**

# Ceph Integration

- Implementing XO-Ceph S3 Object Gateway
  - Offloads connection to OSD that stores the requested object
  - Offload target driven by object location
  - Improves bandwidth usage and data locality
  - Implements hybrid eBPF+TC-HW offload

17% better with hybrid rule insertion



# Conclusion

- Best of both worlds between L4 and L7 load balancing
  - Connections does not permanently stays at backend
  - No L7 relaying at frontend
- Made TCP connection offload practical
  - Designed offload protocol
  - Eliminate need for for special hardware
  - Takes advantage of fast eBPF map update, and opportunistically use HW offload
- Real World Applications
  - Nginx
  - Ceph

Please reach out for new use cases and application ♡