StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs

Kenichi Yasukata (Keio University*), **Michio Honda**, Douglas Santry, Lars Eggert (NetApp)

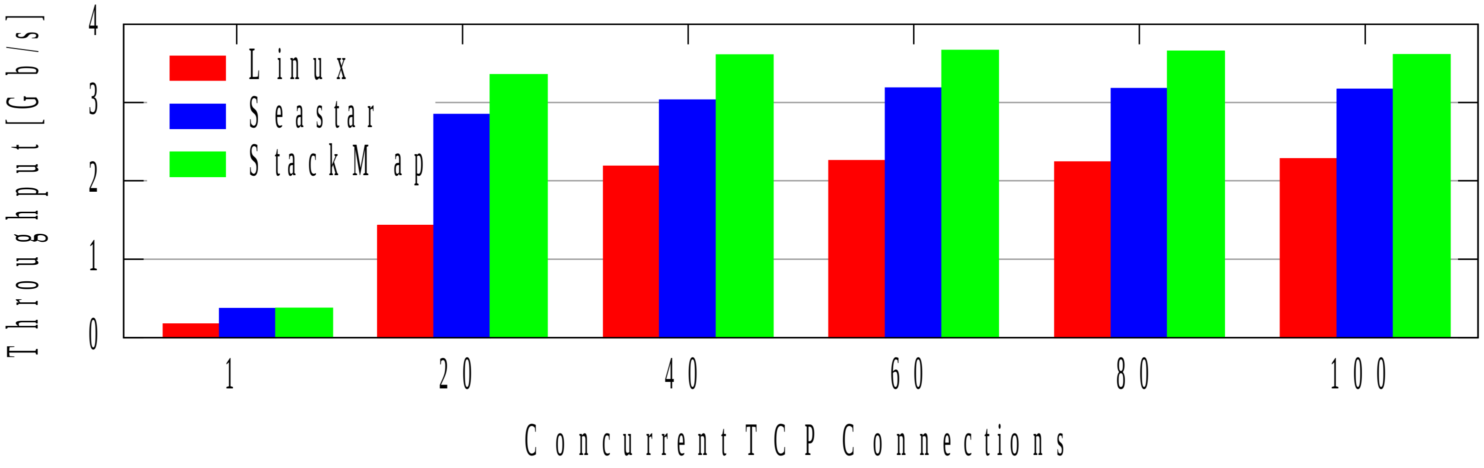
July 18nd @ IETF TSVAREA

*Work while an intern at NetApp

Overview

- Message-oriented communication over TCP is common
  - e.g., HTTP, memcached, CDNs
- Linux network stack can serve 1KB messages only at 3.5 Gbps w/ a single core
- Improve socket API?
  - Limited Improvements
- User-space TCP/IP stack?
  - Maintaining and updating today’s complex TCP is hard

StackMap achieves high performance with the OS TCP/IP

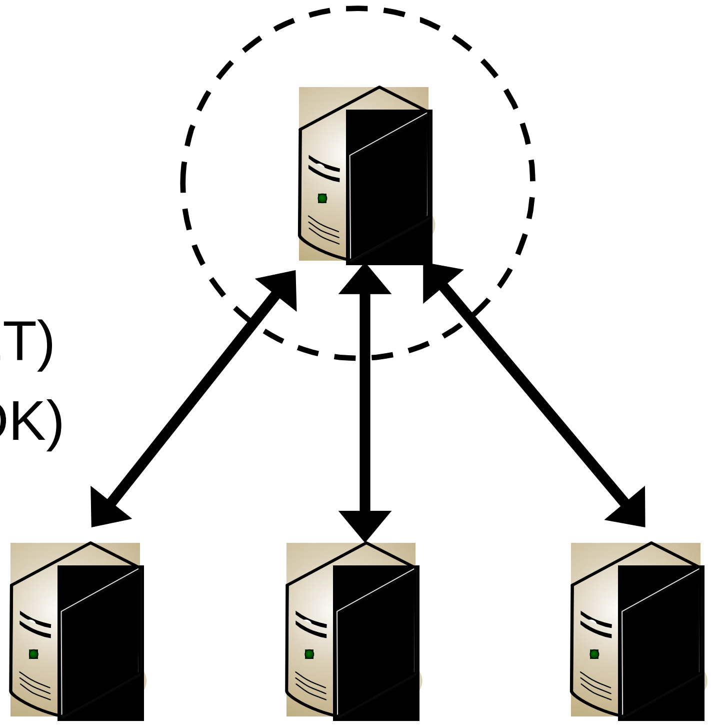


NetApp

Background

- Message-oriented communication over TCP (e.g., HTTP, memcached)
  - Many concurrent connections
  - Small messages
  - High packet rates

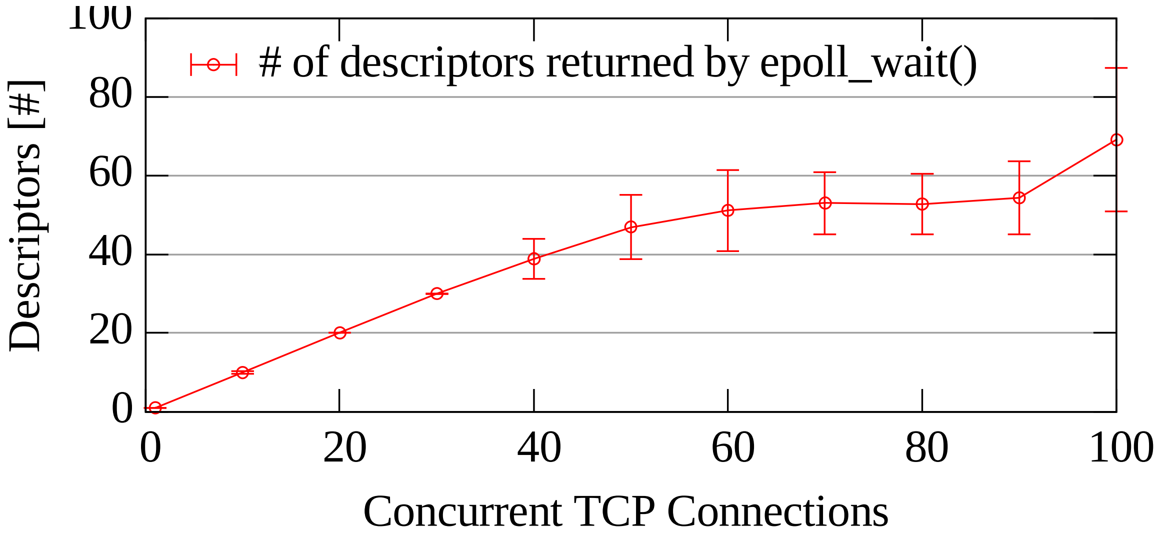
Request (e.g., HTTP GET)
Response (e.g., HTTP OK)

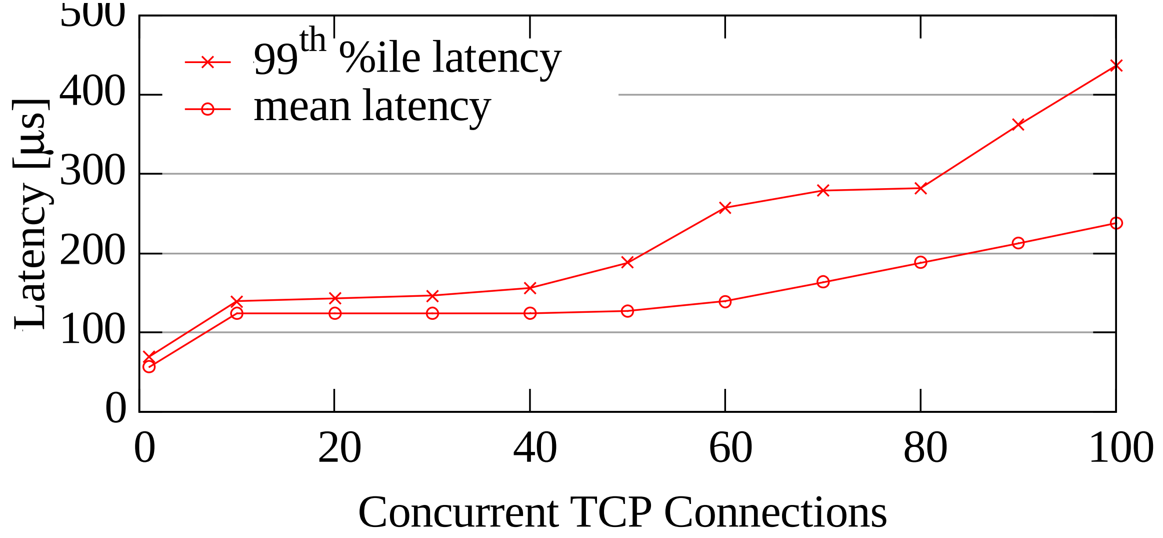


NetApp

Message Latency Problem

```
while (1) {
  n = epoll_wait(fds);
  for (i = 0; i < n; i++) {
    read(fds[i], buf)
    http_ok(buf);
    write(fds[i], buf);
  }
}
```



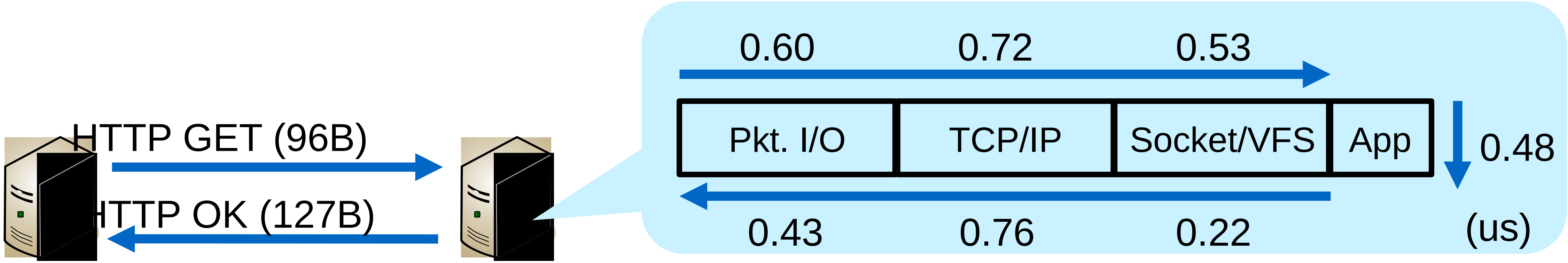


- Many requests are processed in each epoll_wait() cycle

NetApp

Where Could We Improve?

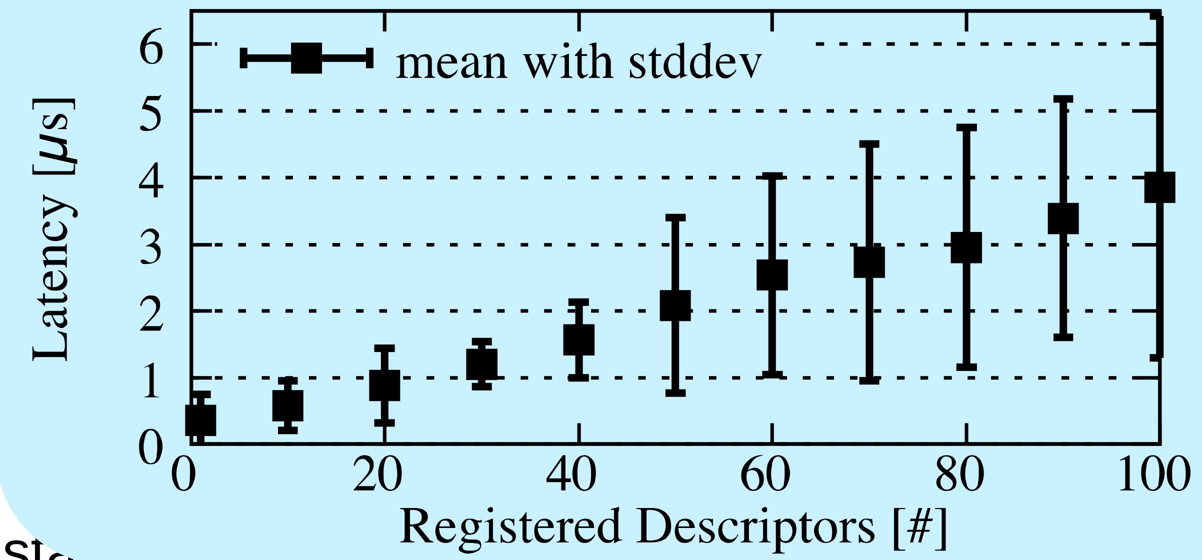
- Processing cost of **TCP/IP protocol** is not high
- TCP/IP takes 1.48 us, out of 3.75 us server processing
- ½ RTT reported by the client app is 9.75 us
  - The rest of 6 us come from minimum hard/soft indirection
  - netmap-based ping-pong (network stack bypass) reports 5.77 us

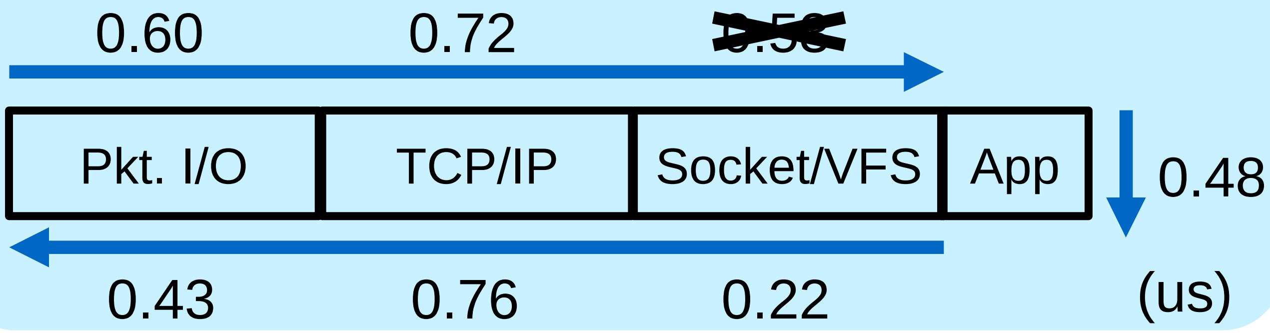


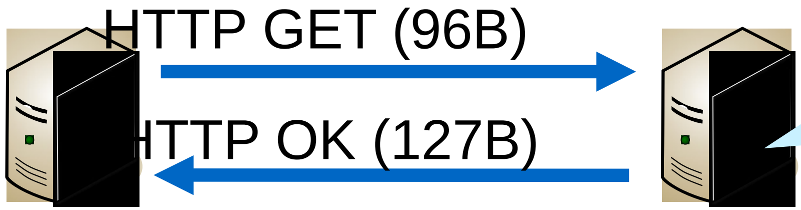
NetApp

Where Could We Improve?

- Processing cost of **TCP/IP protoco**
- TCP/IP takes 1.48 us, out of 3.75 u
- ½ RTT reported by the client app is
  - The rest of 6 us come from minimu
  - netmap-based ping-pong (network sta







Takeaway

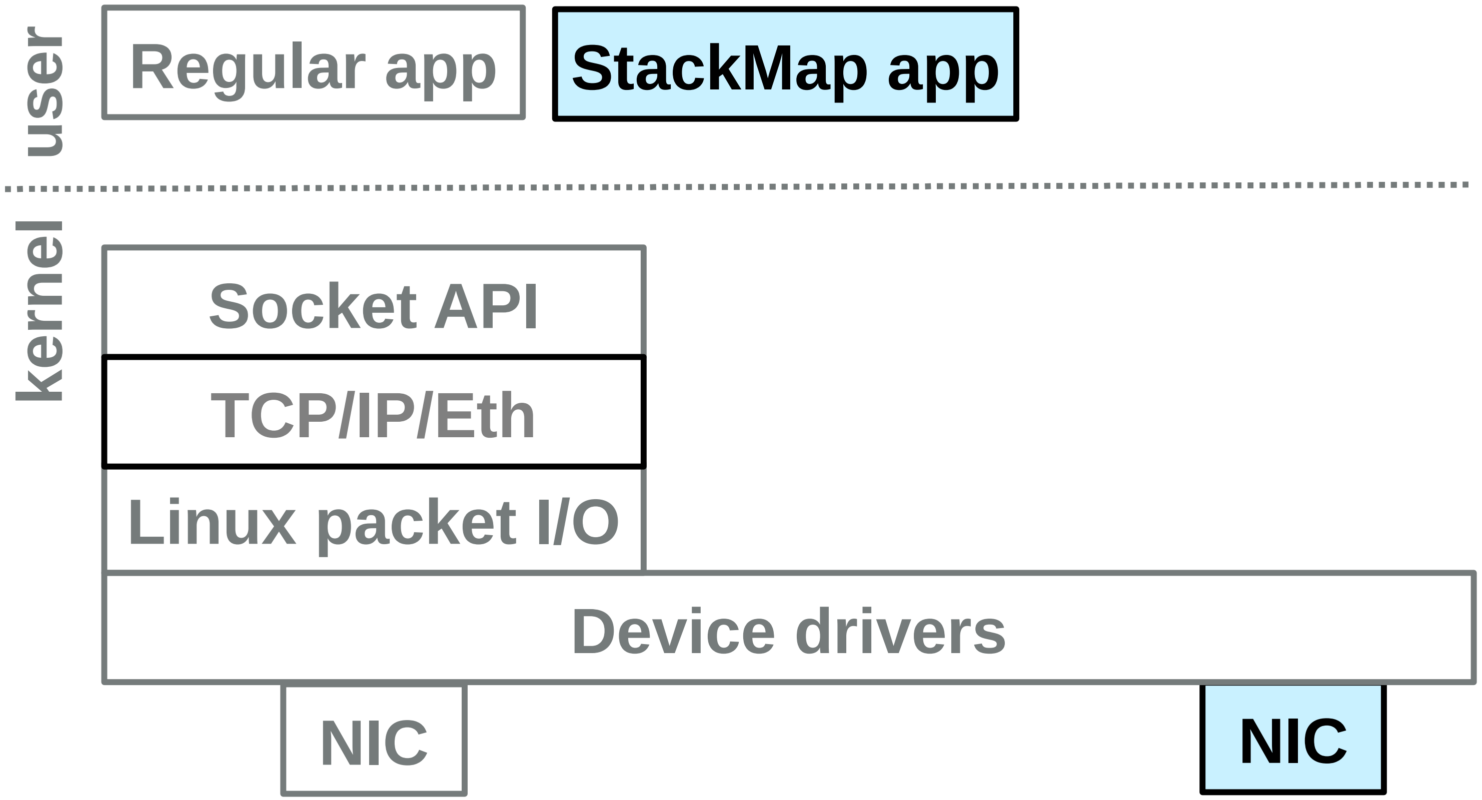
- **Conventional system introduces end-to-end latency of 10’s to 100’s of us**
  - Results of processing delays
- **Socket API comes at a significant cost**
  - read()/write()/epoll_wait() processing delay
- **Packet I/O is expensive**
- **TCP/IP protocol processing is relatively cheap**

We can use the feature-rich kernel TCP/IP implementation, but need to improve API and packet I/O

NetApp

StackMap Approach

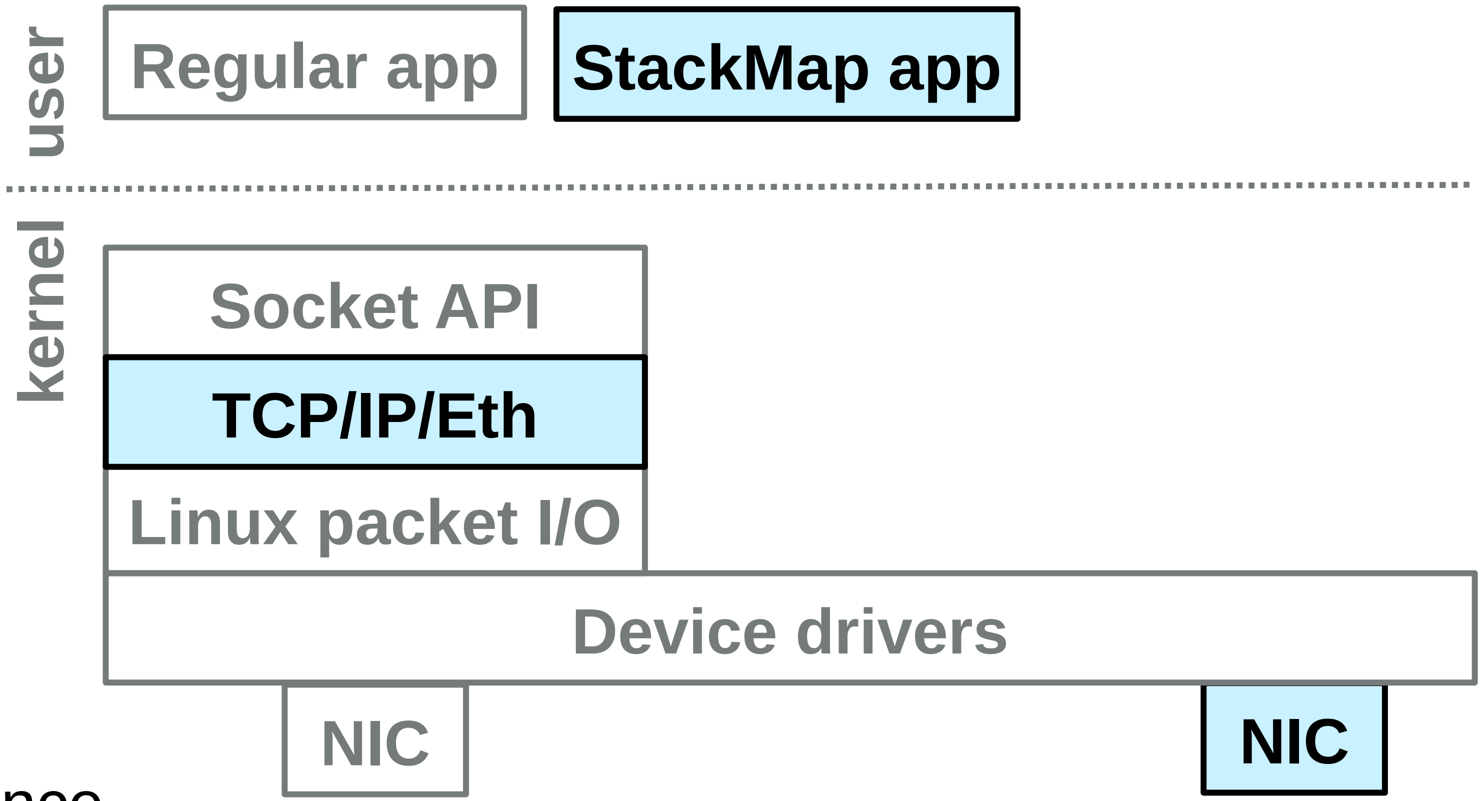
- Dedicating a NIC to an application
  - Common for today's high-performance systems
  - Similar to OS-bypass TCP/IPs



NetApp

StackMap Approach

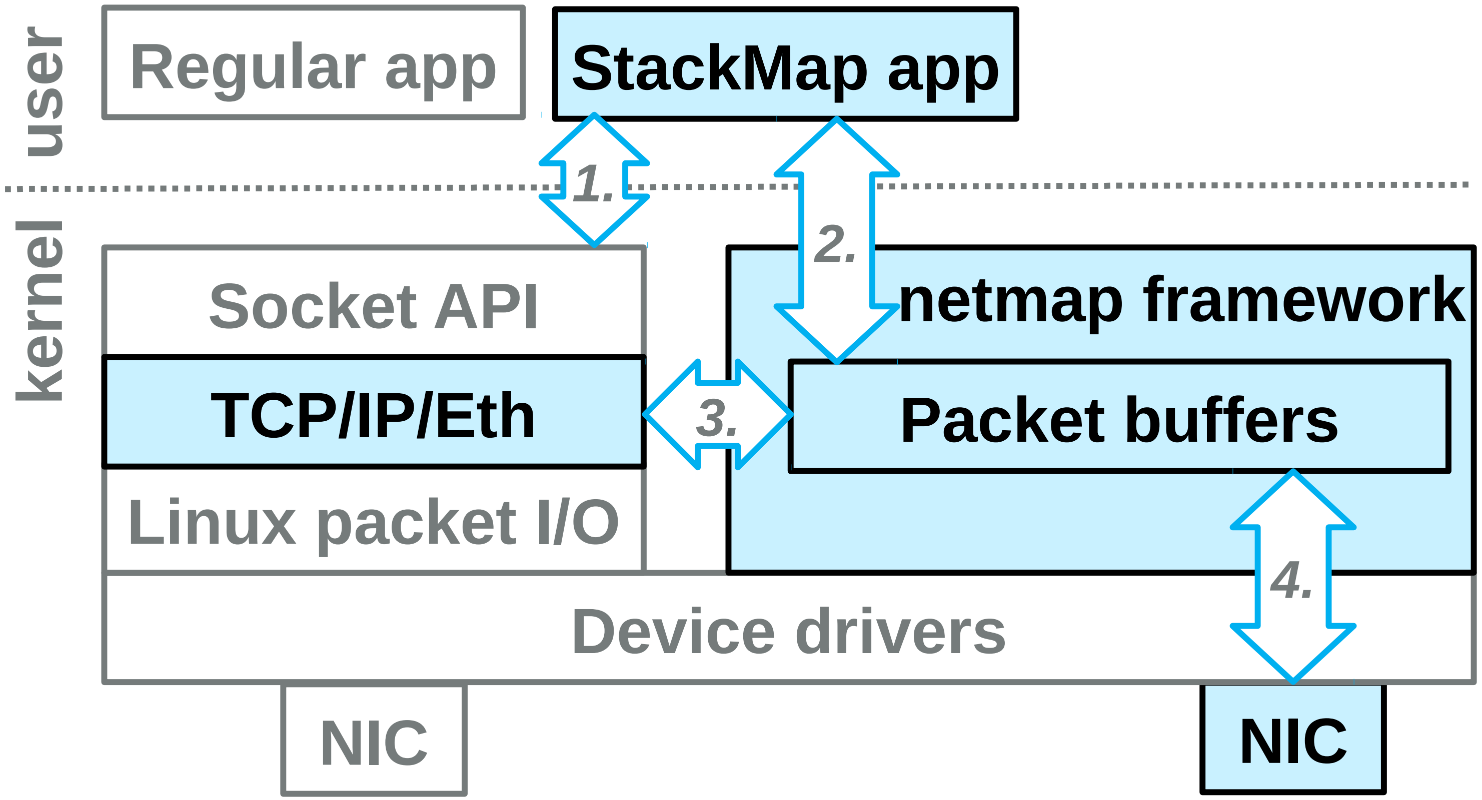
- Dedicating a NIC to an application
  - Common for today's high-performance systems
  - Similar to OS-bypass TCP/IPs
- TCP/IP stack in the kernel
  - State-of-the-art features
  - Active updates and maintenance



NetApp

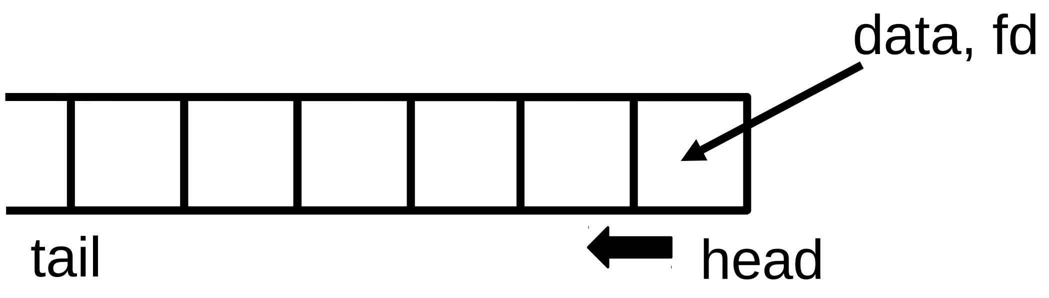
StackMap Architecture

1. Socket API for control path
   - socket(), bind(), listen()
2. Netmap API for data path (extended)
   - Syscall and packet I/O batching, zero copy, run-to-completion
3. Persistent, fixed-size sk_buffs
   - Efficiently call into kernel TCP/IP
4. Static packet buffers and DMA mapping



StackMap Data Path API

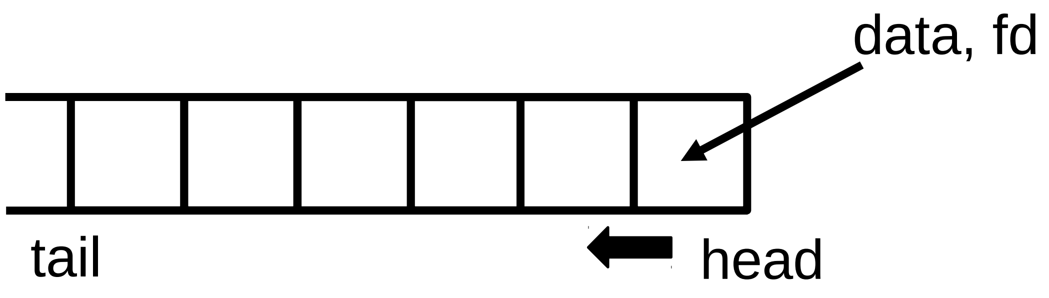
- TX
  - Put data and fd in each slot
  - Advance the head pointer
  - Syscall to start network stack processing and transmission



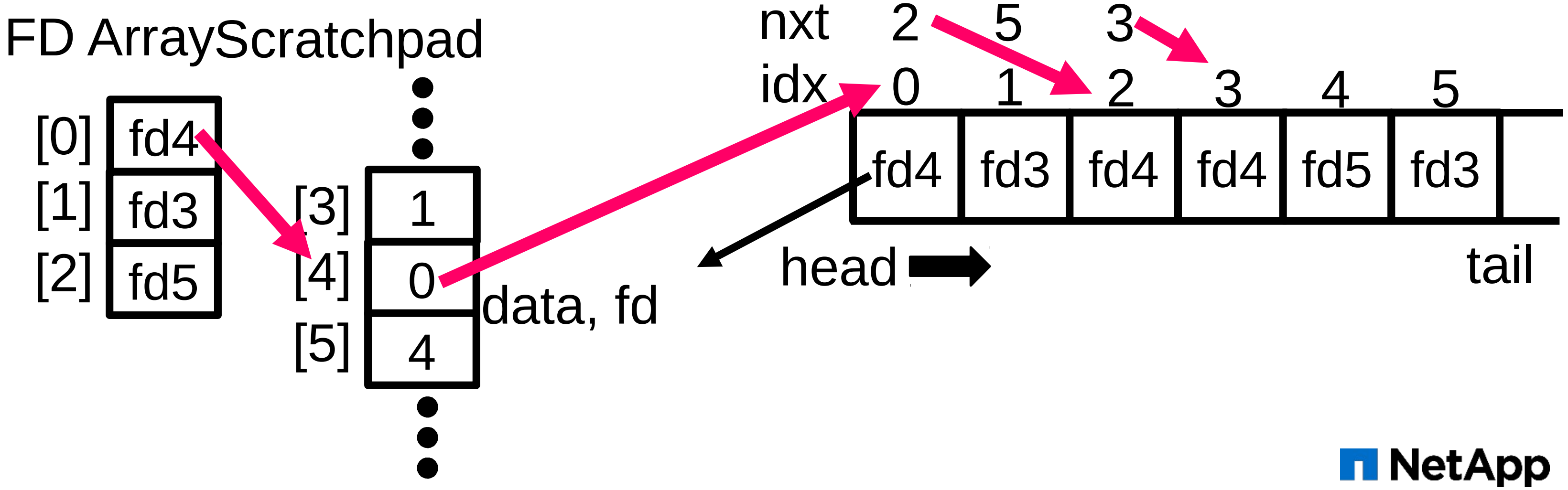
NetApp

StackMap Data Path API

- TX
  - Put data and fd in each slot
  - Advance the head pointer
  - Syscall to start network stack processing and transmission



- RX
  - Kernel puts fd on each buffer
  - App can traverse a ring by descriptors



NetApp

Experimental Results

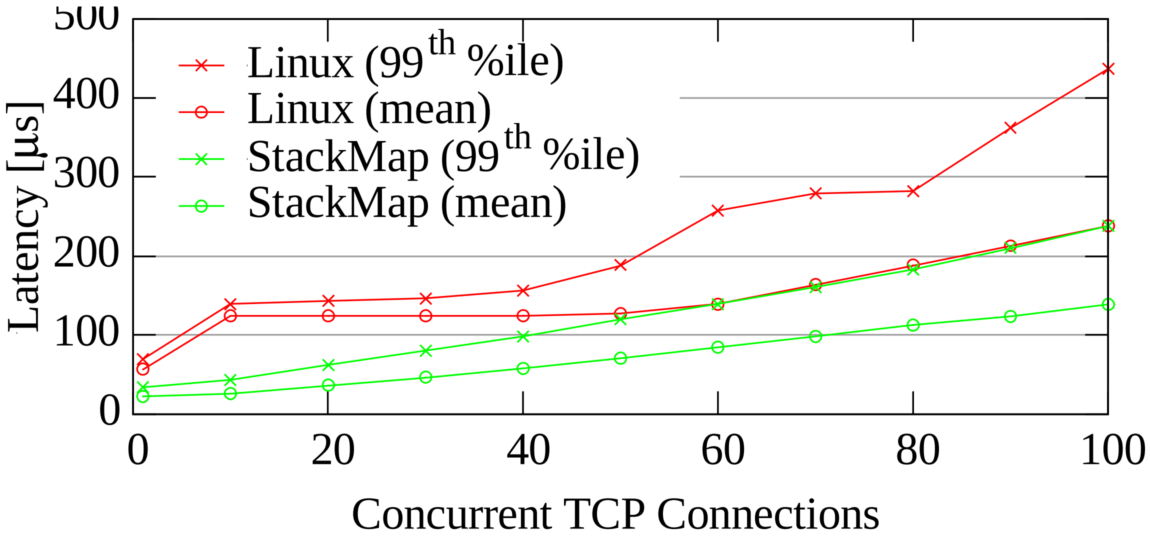
- Implementation
  - Linux 4.2 with 228 LoC changes
  - netmap with 56 LoC changes
  - A new kernel module with 2269 LoC

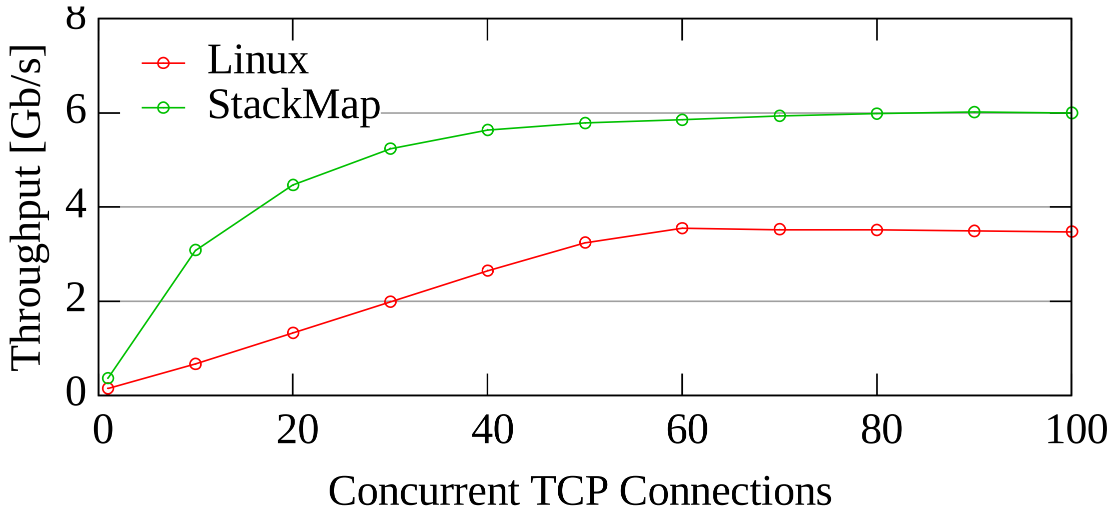
- Setup
  - Two machines with Xeon E5-2680 v2 (2.8 -3.6 Ghz) Intel 82599 10 GbE NIC
  - Server: Linux or StackMap
  - Client: Linux with **WRK** http benchmark tool or **memaslap** memcached benchmark tool

NetApp

Basic Performance

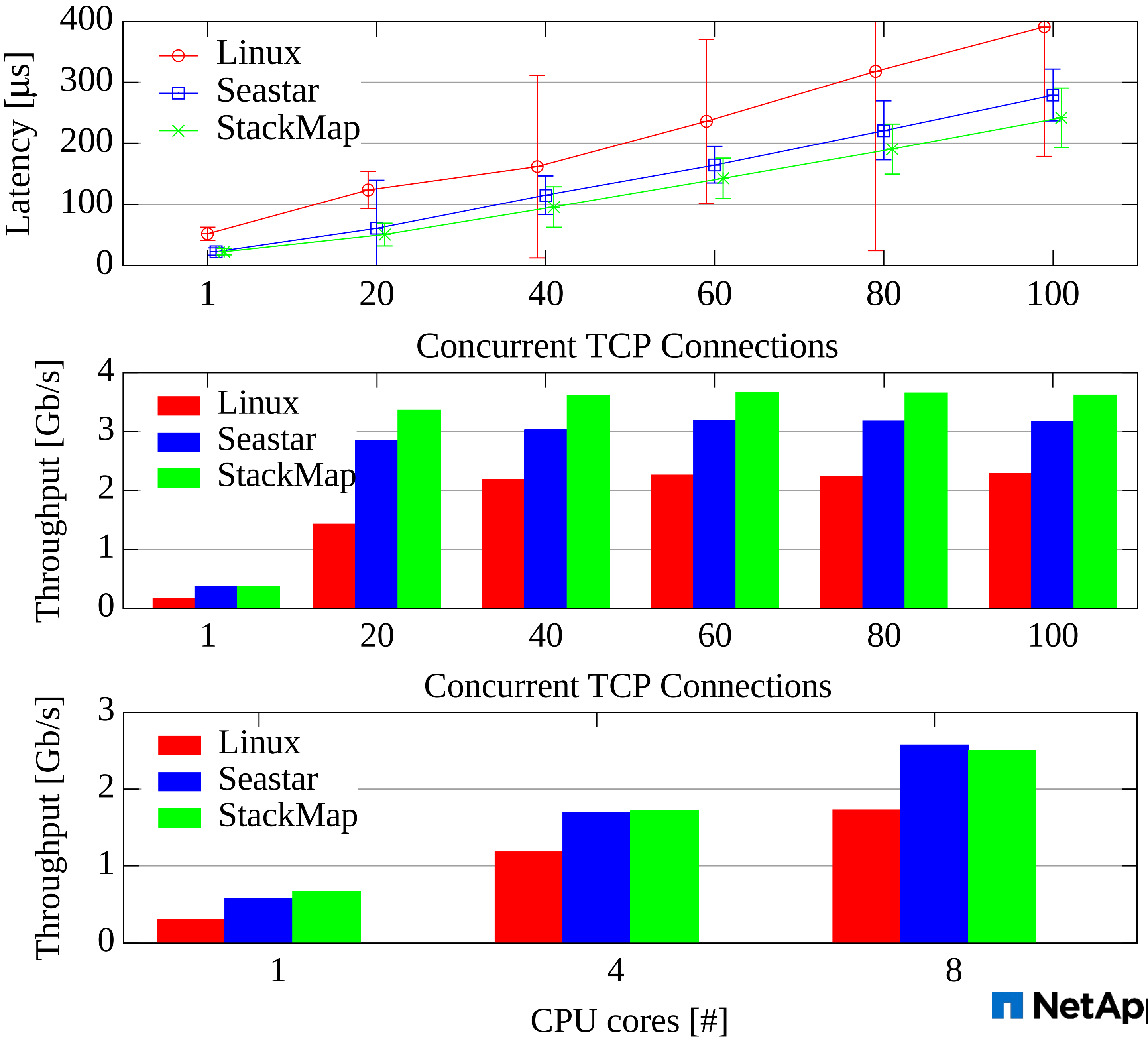
- Simple HTTP server
  - Serving 1KB messages (single core)





Memcached Performance

- Serving 1KB messages
  - single core
  - Seastar is a fast user-space TCP/IP on on top of DPDK*
- Serving 64B messages
  - 1-8 CPU cores



*http://www.seastar-project.org/

Discussion

- What makes StackMap fast?
  - Techniques used by OS-bypass TCP/IPs
    - Run-to-completion, static packet buffers, zero copy, syscall and I/O batching and new API
- Limitations and Future Work
  - Safely sharing packet buffers
    - If kernel-owned buffers are modified by a misbehaving app, TCP might fall into inconsistent state

NetApp

Related Work

- Kernel-bypass TCP/IPs
  - IX [OSDI’14], Arrakis [OSDI’14], UTCP [CCR’14], Sandstorm [SIGCOMM’14], mTCP [NSDI’14], Seastar
- Socket API enhancements
  - MegaPipe [OSDI’12], FlexSC [OSDI’10], KCM [Linux]
- Improving OS stack with fast packet I/O
  - mSwitch [SOSR’15]
- In-stack improvement
  - FastSocket [ASPLOS’16]
- Running kernel stack in user-space
  - Rump [AsiaBSDCon’09], NUSE [netdev’15]

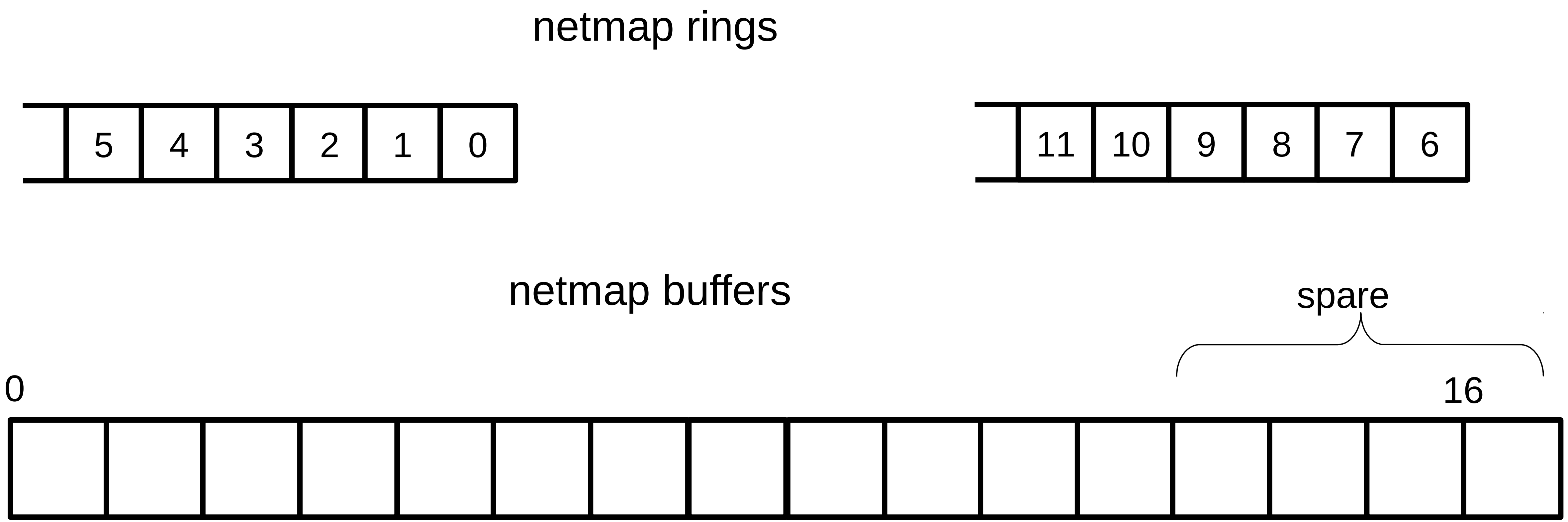
Conclusion

- Message-oriented communication over TCP
- Kernel TCP/IP is fast
  - But socket API and packet I/O are slow
- We can bring the most of techniques used by kernel-bypass stacks into the OS stack
- Latency reduction by 4-80% (average) or 2-70% (99^{th}%tile)
- Throughput improvement by 4-391%

How netmap accelerate the OS stack?

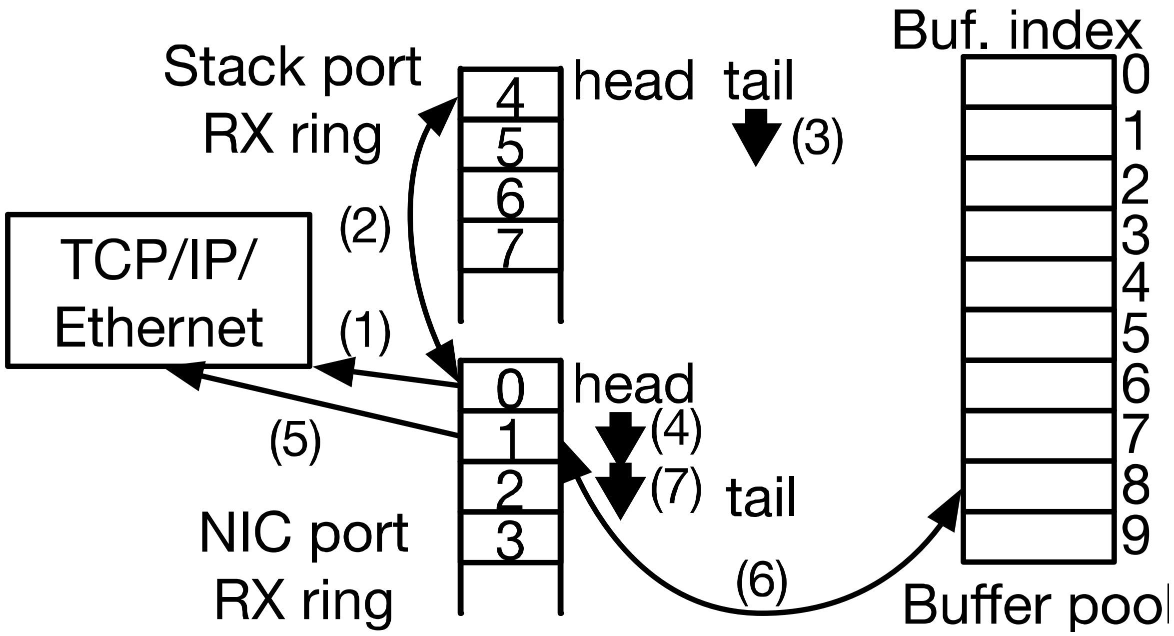
- netmap + Open vSwitch kernel datapath (with VALE/mSwitch)
  - 3x (3.2 Mpps)
- netmap + FreeBSD IP routing (w/ DXR)
  - 1.6x (2.6Mpps)
- netmap + Linux TCP/IP (i.e., Stackmap)
  - 3.5x (787Mbps) (for 64B messages)
  - Replace socket API in addition to packet I/O

netmap rings/slots and buffers



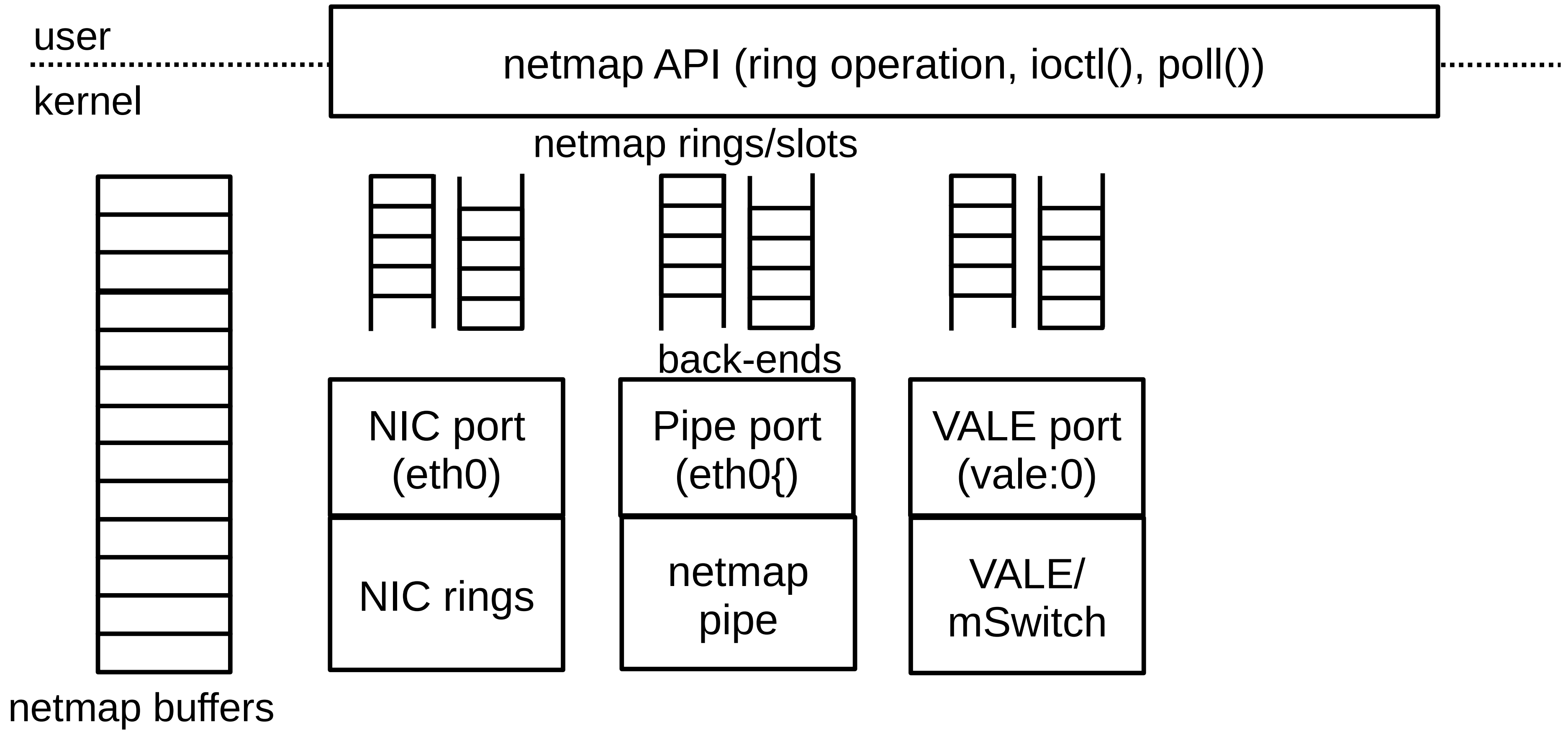
Stackmap Data Path

- On RX, packets are moved from the NIC port ring to a Stack port ring
  - Buffers are moved by swapping buffer indices
  - In the picture the second packet is identified as out-of-order
- TX is done in a similar way



NetApp

Netmap Framework Overview



NetApp

Netmap Framework Overview