

Host Network Stack Overheads

ECE/CS598HPN

Radhika Mittal

First Progress Report

- Original deadline was Friday, Oct 9, 11:59pm.
- Deadline extended to Monday, Oct 12, 11:59pm.

P4 usecases (in-network mechanisms)

- Network-level load balancing:
 - HULA: Scalable Load Balancing Using Programmable Data Planes, SOSR'16
- Congestion control:
 - Evaluating the Power of Flexible Packet Processing for Network Resource Allocation, NSDI'17
 - *Support RCP and XCP on programmable switches*
 - HPCC: High Precision Congestion Control, SIGCOMM'19
 - *Obtain precise link information for congestion control*
- A new protocols for more efficient L2 switching
 - The Deforestation of L2, SIGCOMM'16
- Source-routed multicast
 - Elmo: SIGCOMM'19
- Network monitoring
 - Beaucoup: SIGCOMM'20
 - Marple: SIGCOMM'17

App-level P4 usecases

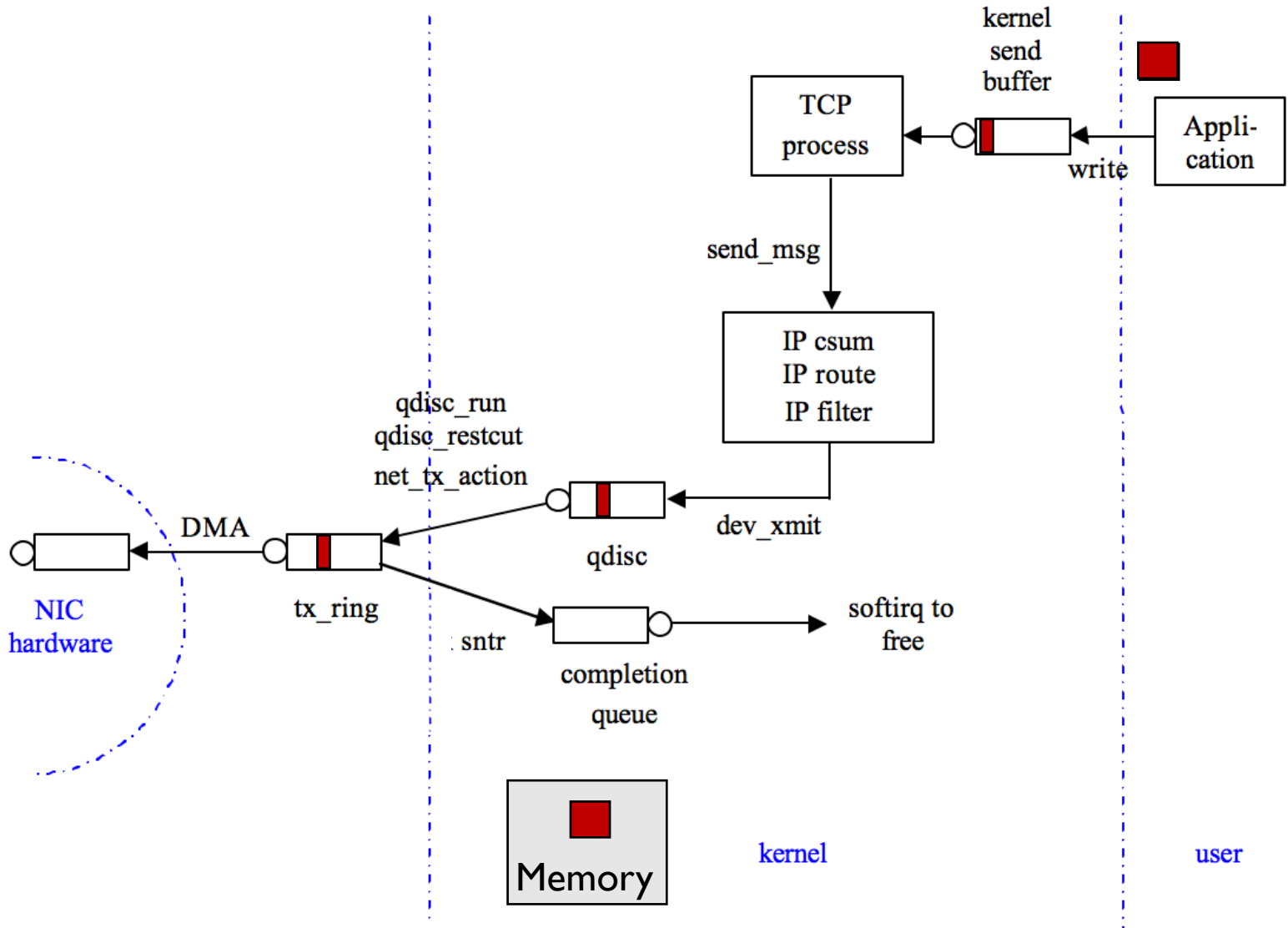
- NetChain: in-network key-value store (NSDI'18).
- NetLock: Switching support to manage locks (SIGCOMM'20).
- NetPaxos: implement Paxos on programmable switches (SOSR'15)
- DAEIT: In-network data aggregation (SOCC'17)
- NoPaxos (OSDI'16), Eris (SOSP'17): in-network primitives for distributed protocols.
- SilkRoad: L4 load balancer (SIGCOMM'17)
- Robot arm control (NSDI'22)
-

Why discontinued?

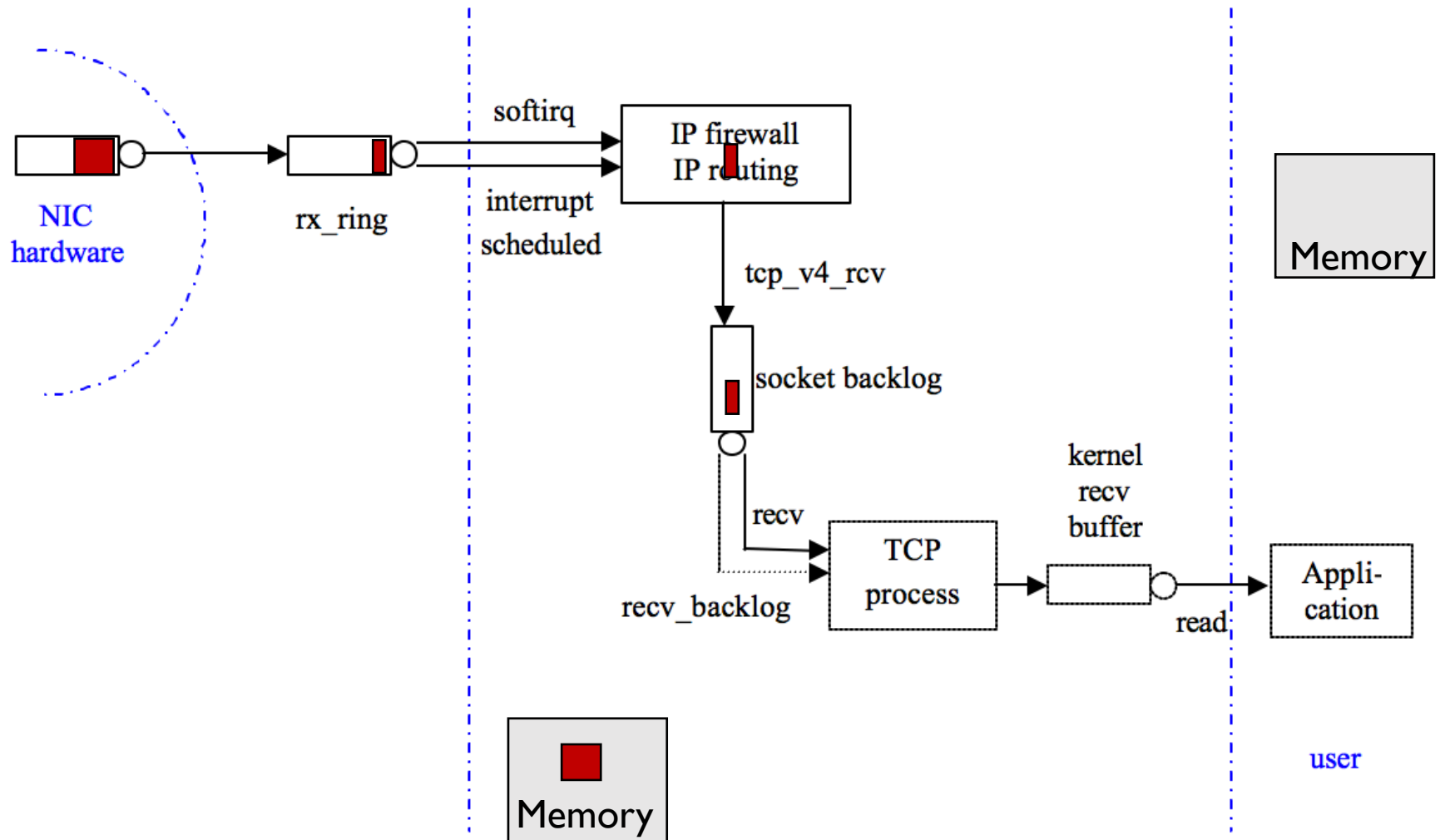
Today's reading

- Understanding Host Network Stack Overheads
 - Cai et. al., SIGCOMM'21
- (Many figures and materials for today's class taken directly from the paper).

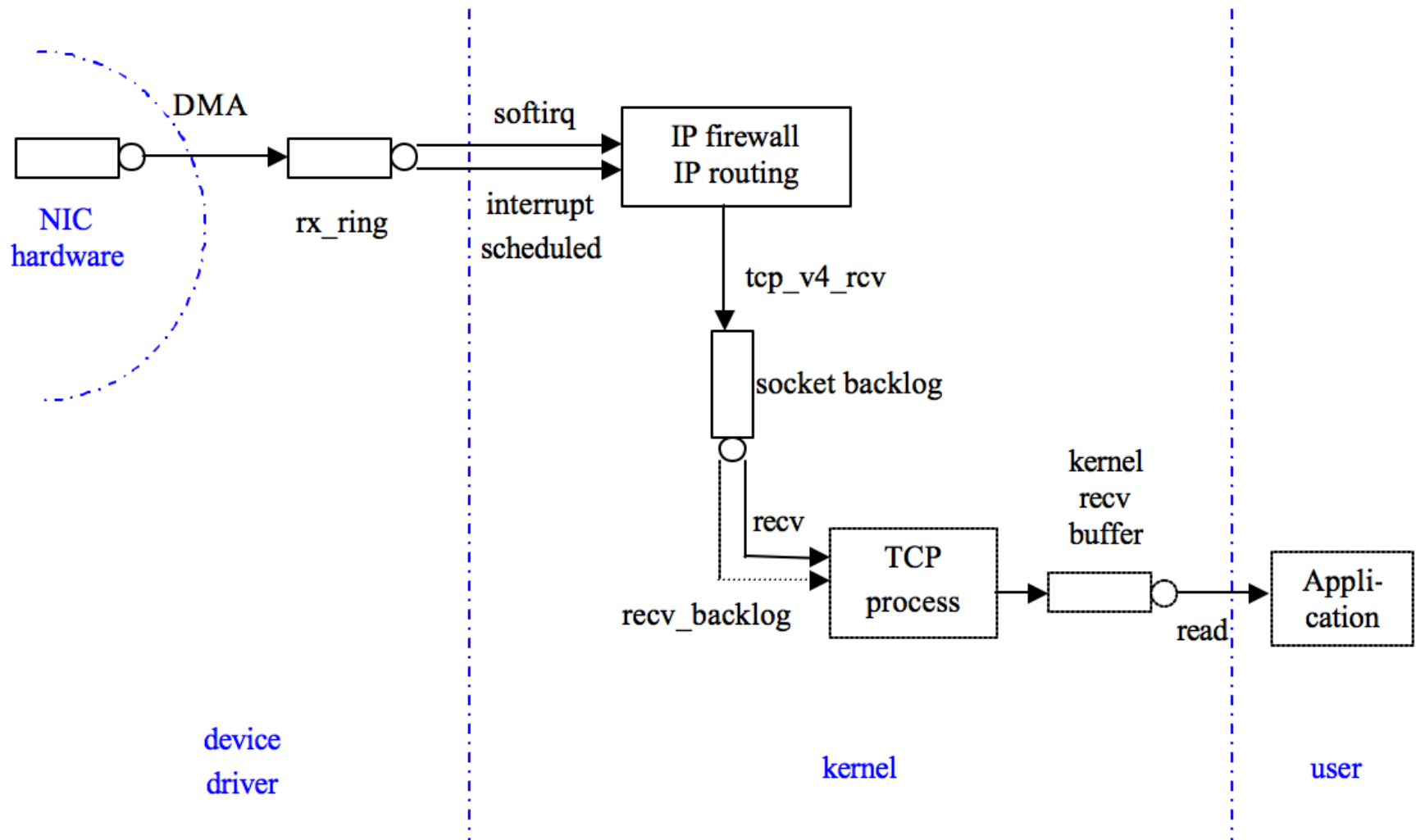
Tx Processing in the kernel



Rx Processing in the kernel



Rx Processing in the kernel



What are some sources of performance overheads?

Sources of overhead

- Protocol processing
- Data copy
- Cache contention (between flows sharing same NUMA node)
- CPU scheduling overheads (locking, context switching)
- Interrupts
- Managing heavy datastructures (skbs)

Optimizations

- NAPI polling:
 - reduces number of interrupts
- DCA/DDIO: Direct Cache Access / Data Direct IO:
 - receiver can DMA packet directly into cache.

Optimizations

- Receive Packet steering / Receive Flow Steering
 - Steering packets to specific queues (cores)
 - Can be done in software / NIC.

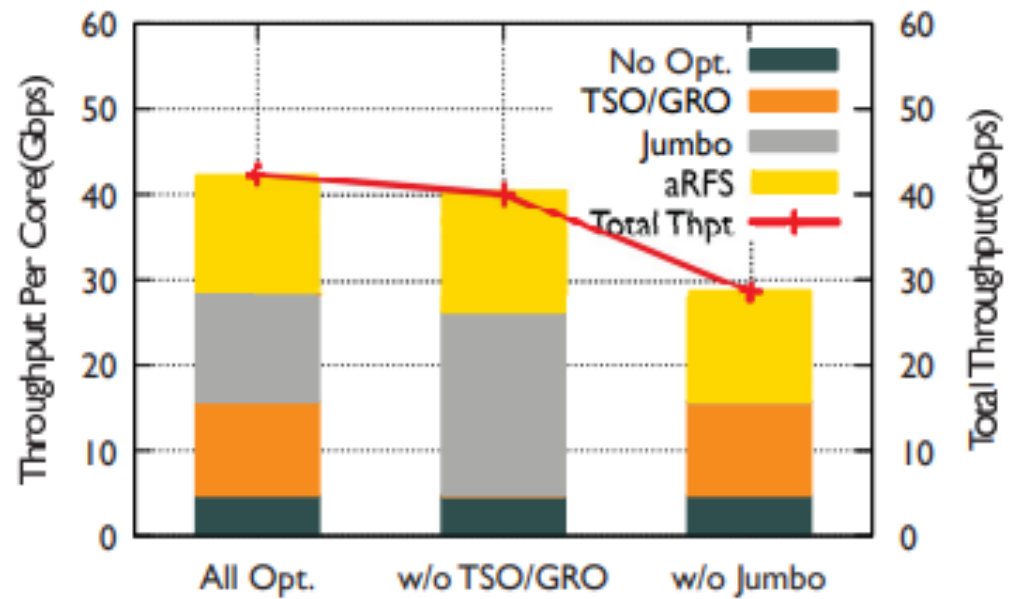
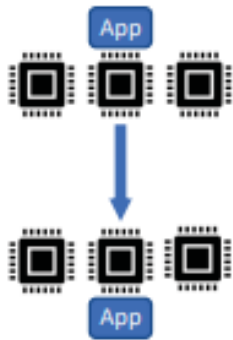
Mechanism	Description
Receive Packet Steering (RPS)	Use the 4-tuple hash for core selection.
Receive Flow Steering (RFS)	Find the core that the application is running on.
Receive Side Steering (RSS)	Hardware version of RPS supported by NICs.
accelerated RFS (aRFS)	Hardware version of RFS supported by NICs.

- RPS goal: load-balancing
- RFS goal: reducing CPU context switches

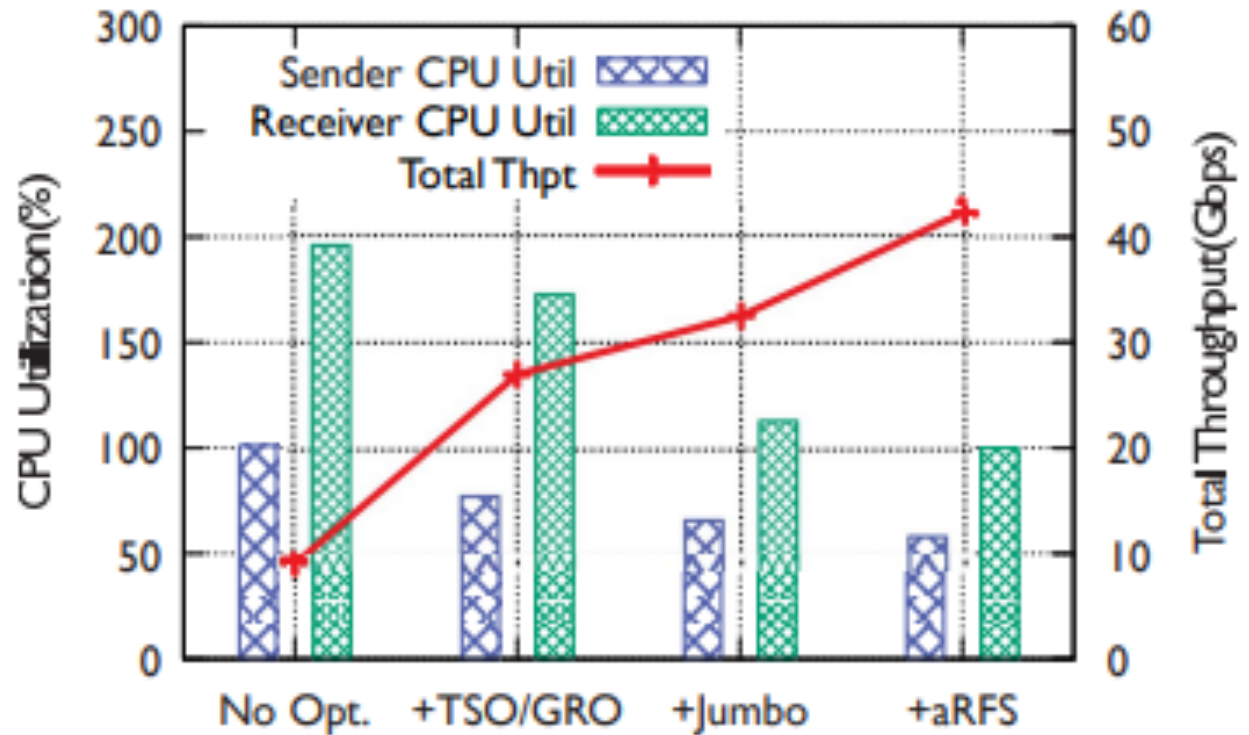
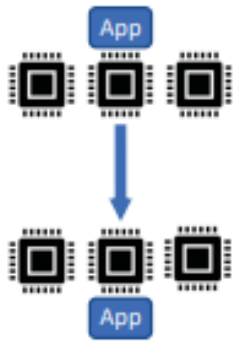
Optimizations

- Segmentation and Receive Offloads
 - Kernel stack processes large 64KB chunks
 - Transmission (tx) side: split chunks into MTU sized (1500 bytes) packets.
 - GSO (generic segmentation offload): splitting in software
 - TSO (TCP segmentation offload): : splitting in NIC
 - Receive (rx) side: aggregate packets into larger chunks
 - GRO (generic receive offload): aggregation in software
 - LRO (large receive offload): aggregation in hardware.
- Jumbo frames: large MTUs (~9000 bytes)
- Zero-copy kernel (not evaluated in the paper)

Results

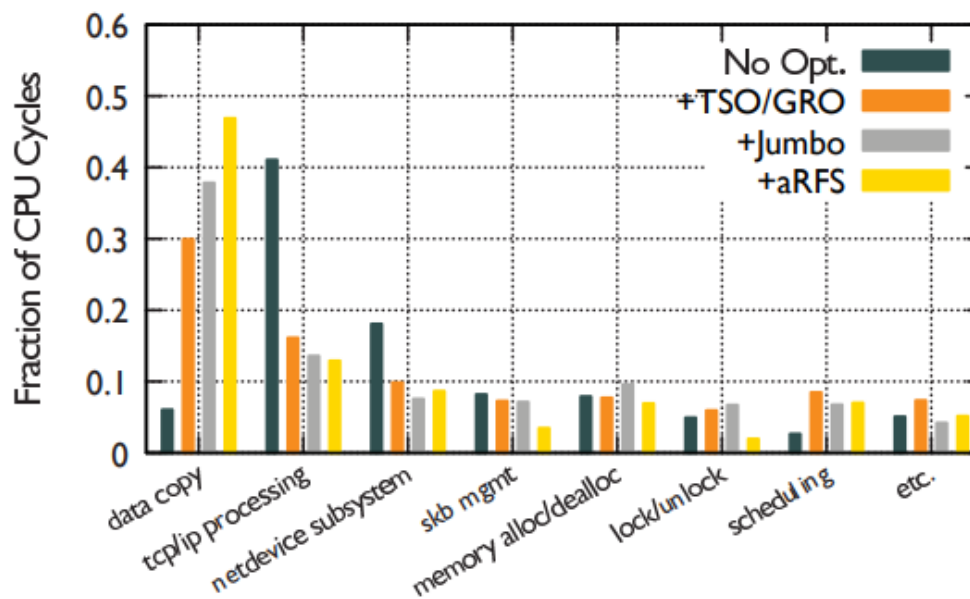
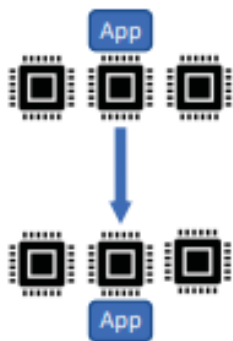


Results



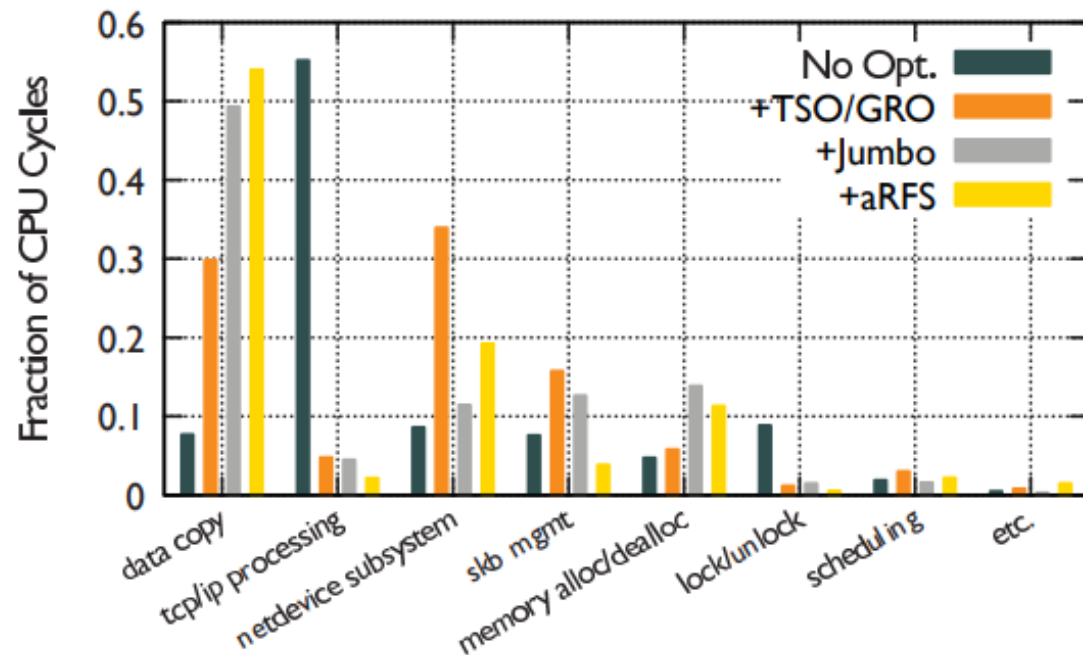
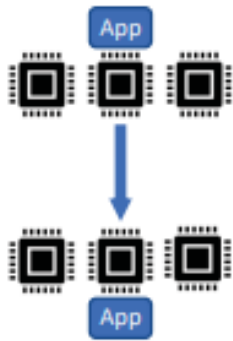
What's the bottleneck?

Results



(c) Sender CPU breakdown

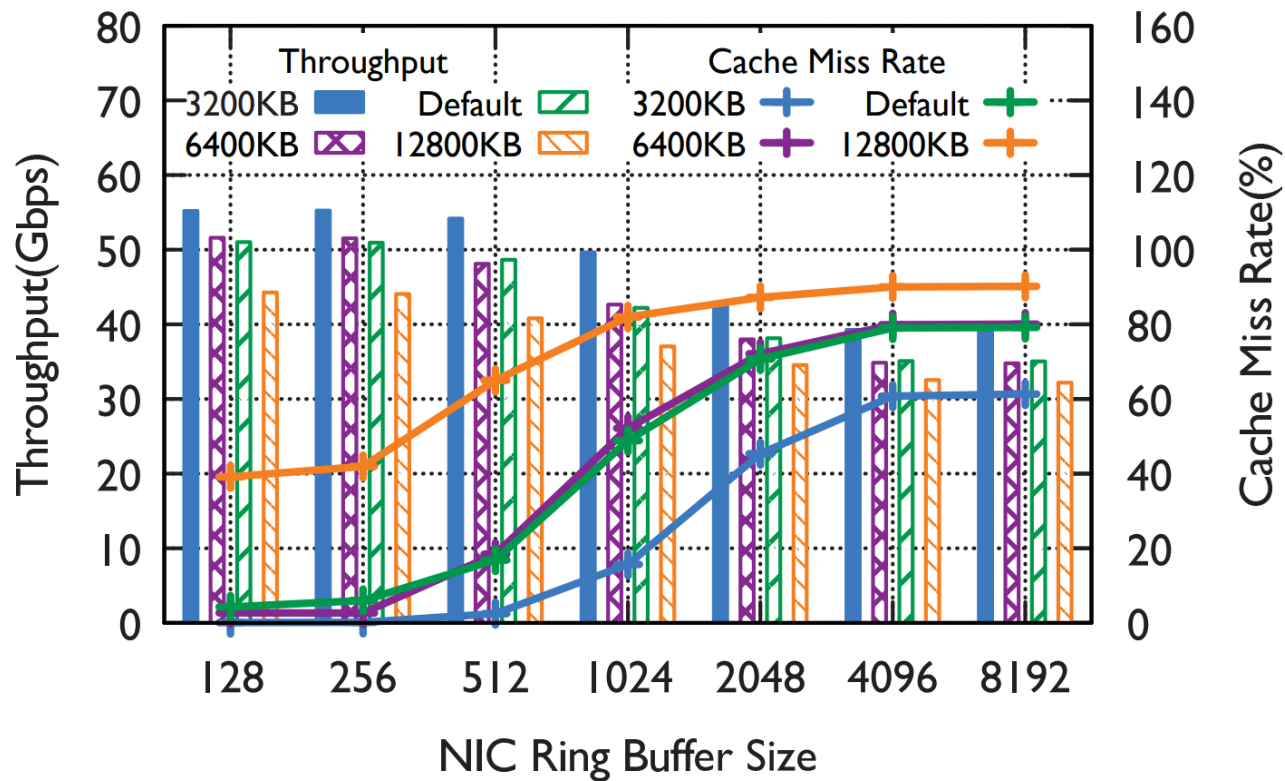
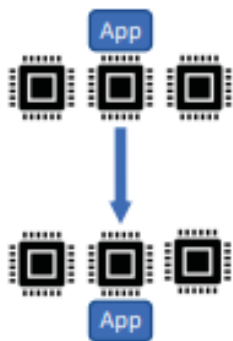
Results



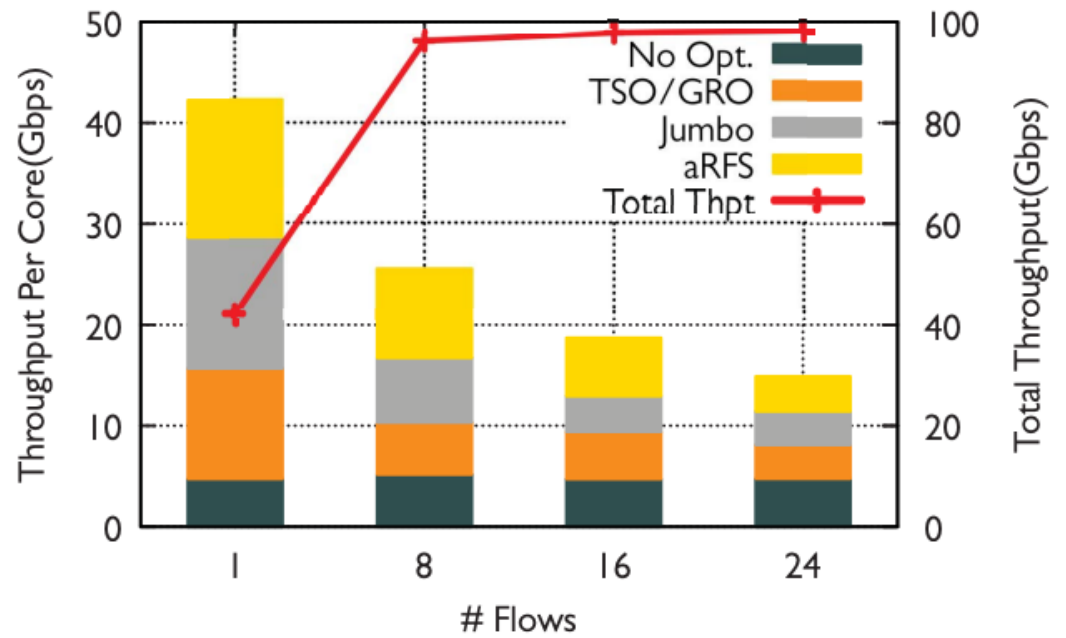
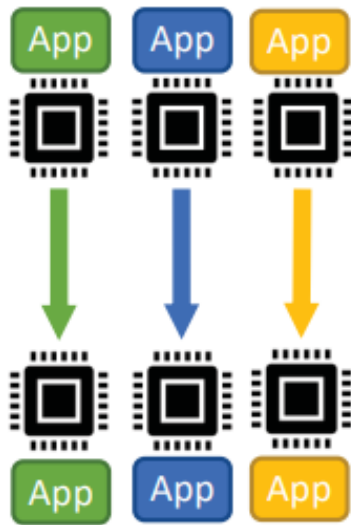
(d) Receiver CPU breakdown

What's the bottleneck if your workload comprises of lots of short flows?

Results

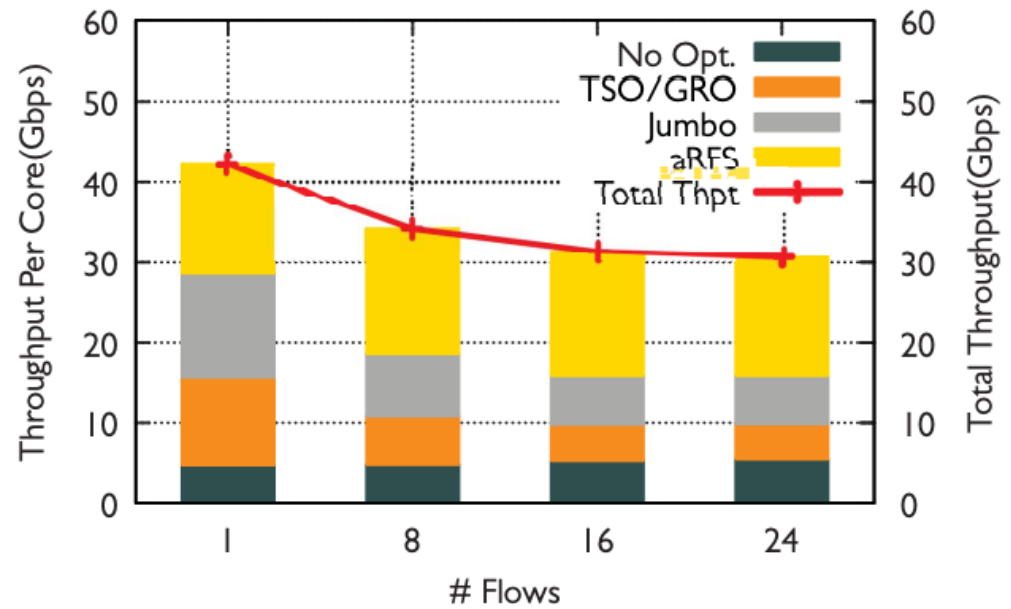


Results



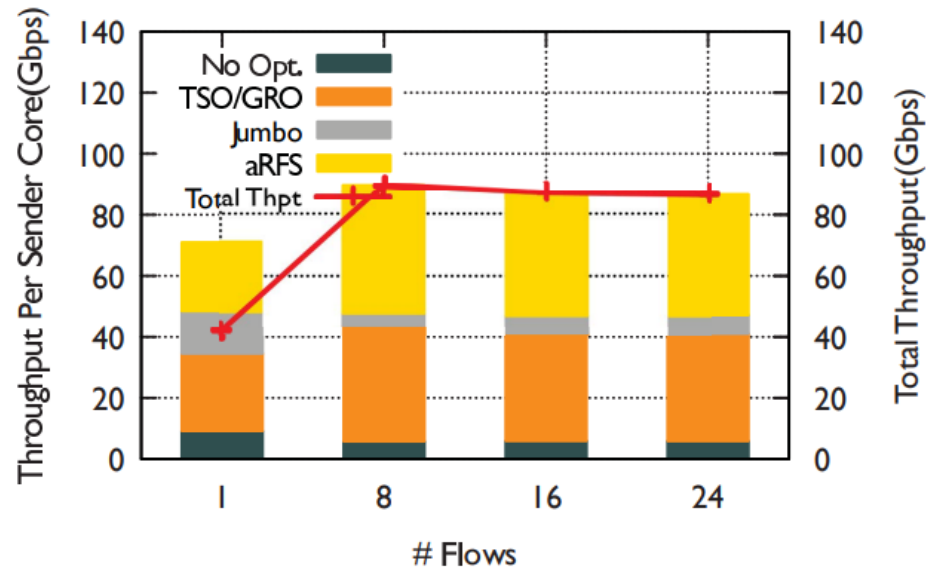
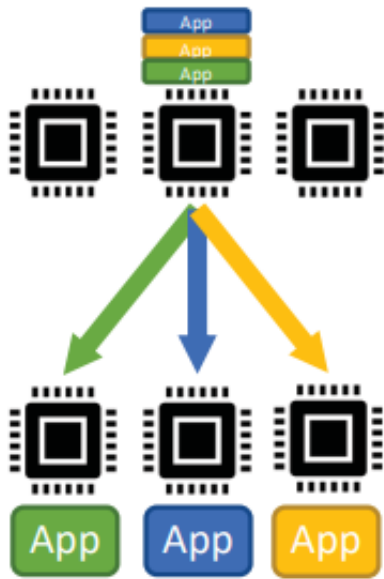
Reasons?

Results



Reasons?

Results

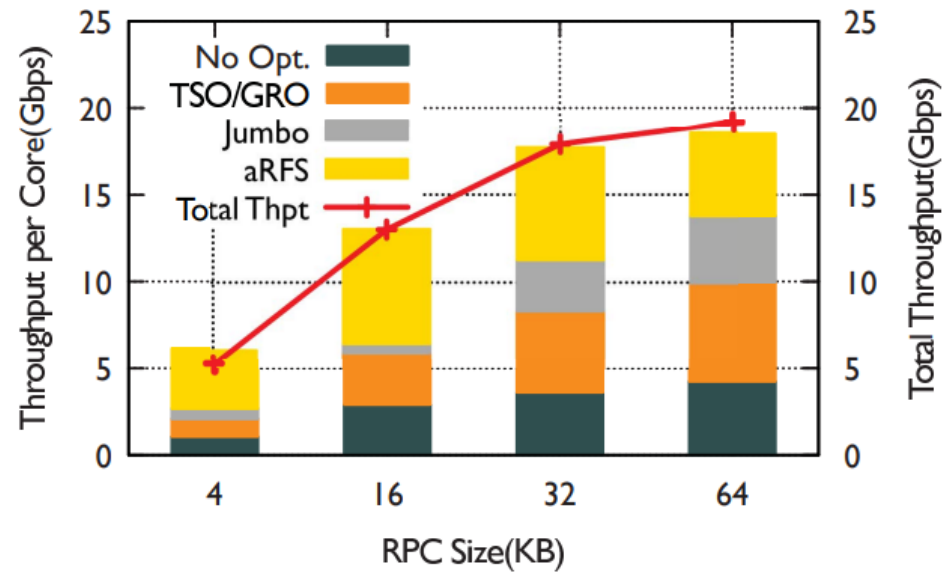


Reasons?

Results



16:1 incast
ping-pong workload
vary flow (RPC) size

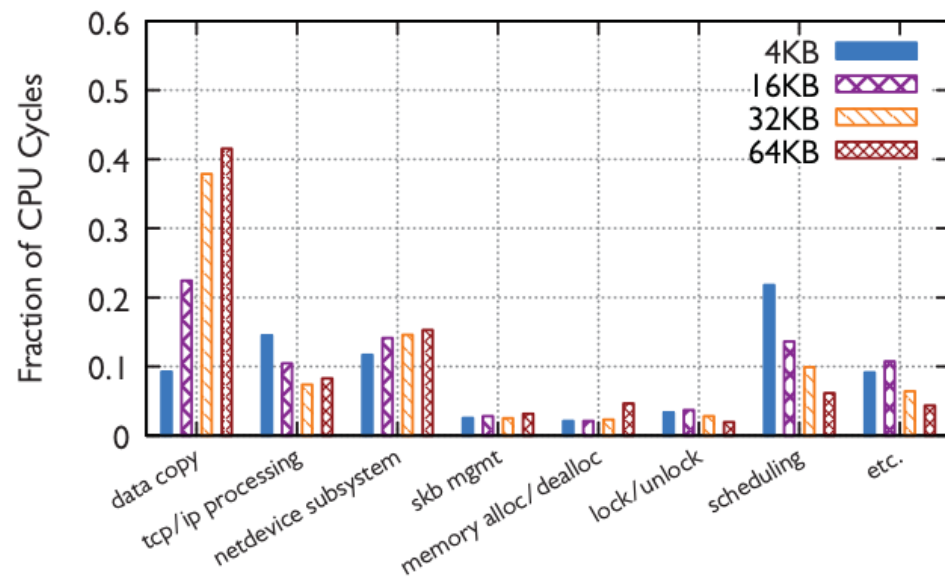


Reasons?

Results



16:1 incast
ping-pong workload
vary flow (RPC) size



Earlier research on kernel optimization

- Megapipe (OSDI'12): proposes new API that supports:
 - batched system calls
 - per-core accept queue (instead of shared accept queue)
 - light-weight sockets (not regular file descriptors)

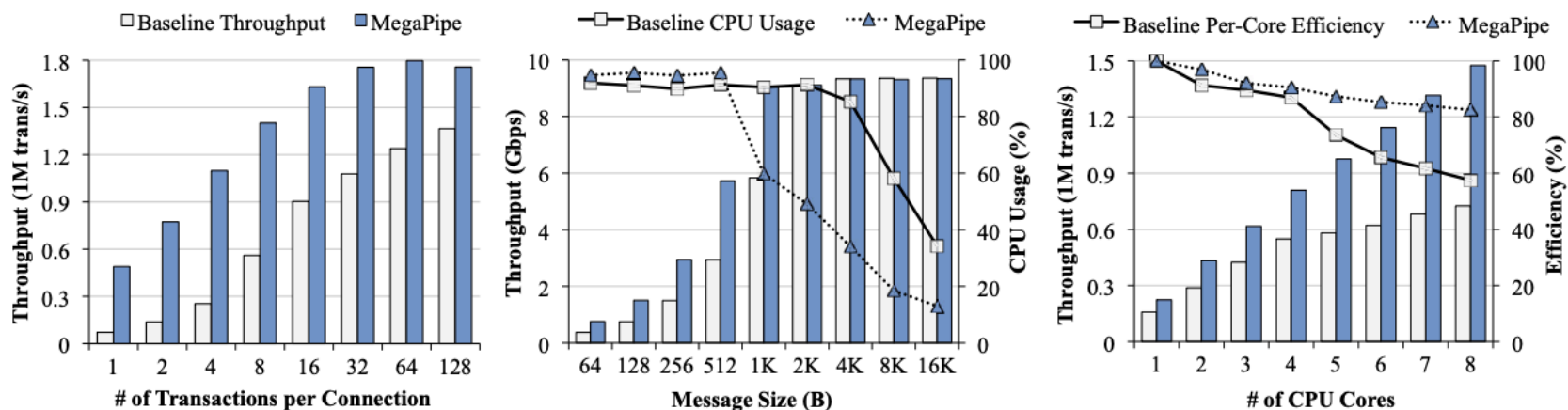


Figure 1: (a) the negative impact of connection lifespan (with 64 B messages on eight cores), (b) message size (with ten transactions per connection on eight cores), and (c) increasing number of cores (with 64 B messages and ten transactions per connection).

Summary

- Short flow bottlenecks:
 - CPU: software packet processing, interrupts, scheduling.
- Long flow bottlenecks (after enabling all optimizations):
 - network: up to 40Gbps
 - data copy in the host, beyond that.
 - zero-copy kernel: implications?

In following classes

- Kernel bypass: userspace network stacks based on DPDK
 - IX: DPDK with run-to-completion and zero-copy API
- Leveraging eBPF for “express datapath”.
- Software bypass and zero-copy
 - RDMA