Google Cloud

Tuning Linux TCP for data-center networks

Yuchung Cheng <ycheng@google.com>

Linux Plumbers Conference
Dublin, Ireland, Sep 2022

Outline

- Datacenter Networks
- TCP network performance
- TCP host performance
- TCP visibility
- The next decade TCP

Who we are

Linux TCP & netdev contributors @ Google Network Infrastructure

- Talal Ahmad
- Mahesh Bandewar
- Willem de Bruijn
- Shakeel Butt
- Neal Cardwell
- Yuchung Cheng
- Eric Dumazet
- Soheil Hassas Yeganeh
- Van Jacobson
- Priyaranjan Jha
- Coco Li
- Arjun Roy
- Stanislav Fomichev
- Mubashir Adnan Qureshi
- Brian Vazquez
- Wei Wang
- Kevin Yang
- Yousuk Seung

Data center networking (DCN)

Network

Many hosts interconnect Gbps links with <msec latency

ECMP to load-balance flows among links

Applications

Scatter & gather: correlated bursts from many hosts

Care about latency, CPU, and memory

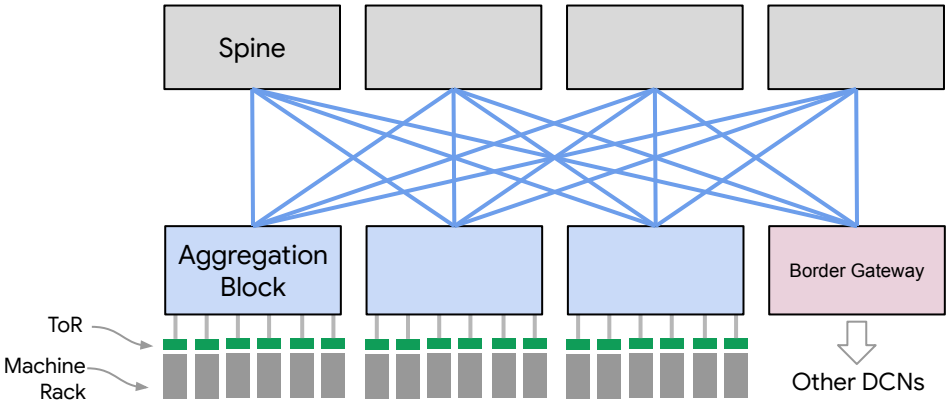
Bottlenecks

Occur in host and networks

But usually invisible to applications

DCN applications' usage critically guides TCP performance tuning

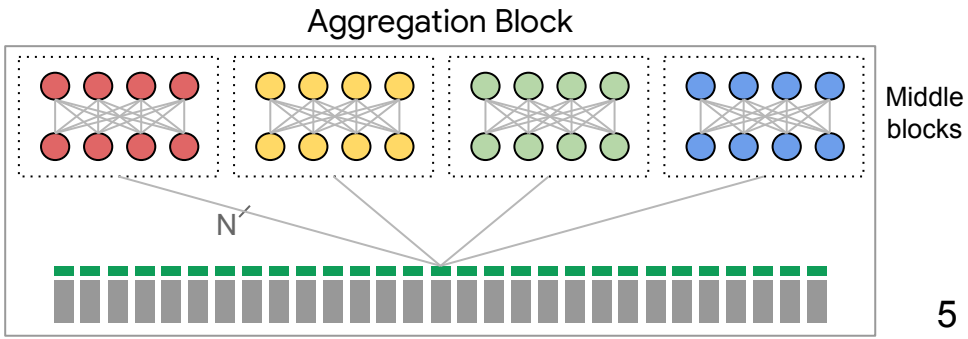
Common DCN 3-tier Clos topology



A host often has many thousands to millions of TCP flows.

An application multiplex messages across them constantly.

A flow idles and bursts.



Optimize TCP network performance

Max BW utilization with minimal queue

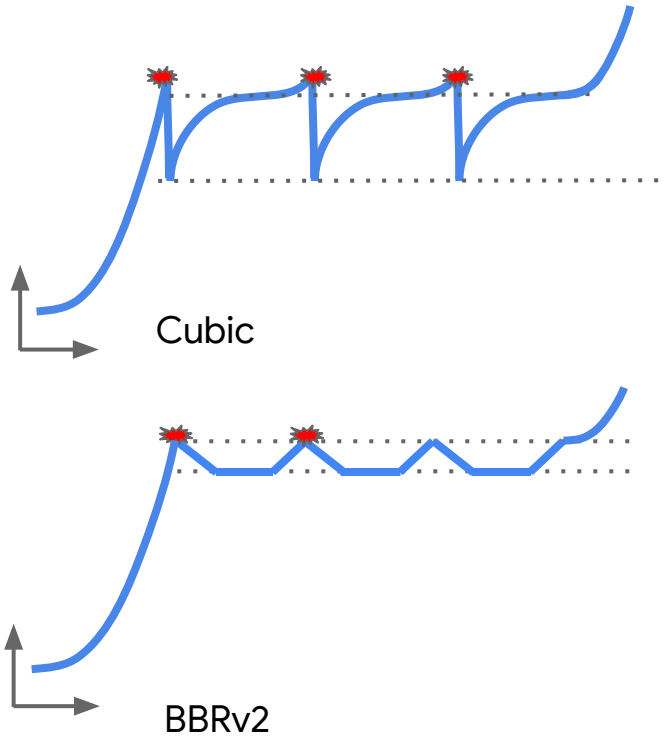
- Use DCTCP or BBRv2 congestion control

Minimize restart overhead

- Use persistent connection and TCP Fast Open
- Disable slow start after idle
  (sysctl net.ipv4.tcp_slow_start_after_idle = 0)

Reduce queuing delays and bursts, improve network entropy

- Replace pfifo with fq/pacing qdisc



Dynamic TCP flowlet to avoid congestion or link failures

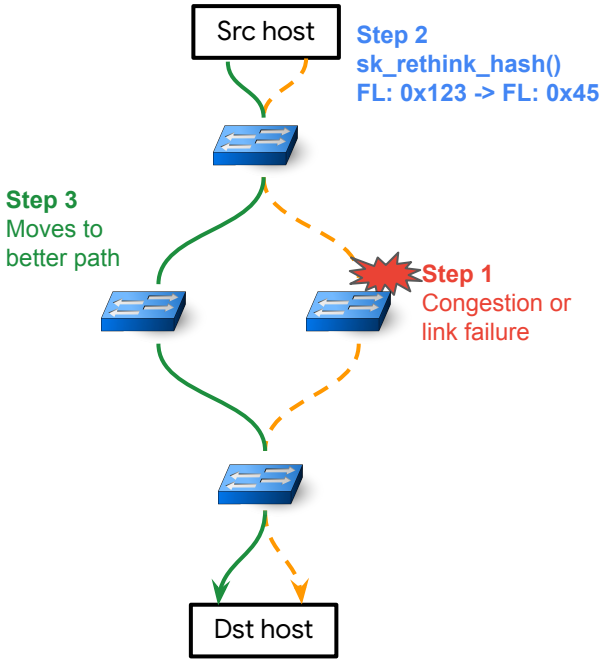
Use Linux TCP feature on IPv6 Flow-Label (RFC6437)

Enable the switches to route with FL in addition to 4-tuples plus ECN

1. TCP flow experiences high ECN [1] or RTO [2] upon congestion or link failures
2. TCP flow re-randomize sk->hash and IPv6 flow-label
3. TCP flow changes its path
4. Repeat 1-3 until no further RTO or ECN

[1] PLB, SIGCOMM 2022

[2]: Default on since kernel 4.x



Tuning TCP WAN performance

- Internet / customer traffic
- Latency sensitive data and controller traffic
- Throughput sensitive data replication

Latency

Reduce the (long) round-trips!

Use persistent connections and disable slow-start after idle

Use RACK-TLP (default) to maximize Fast Recovery instead of RTO

Throughput

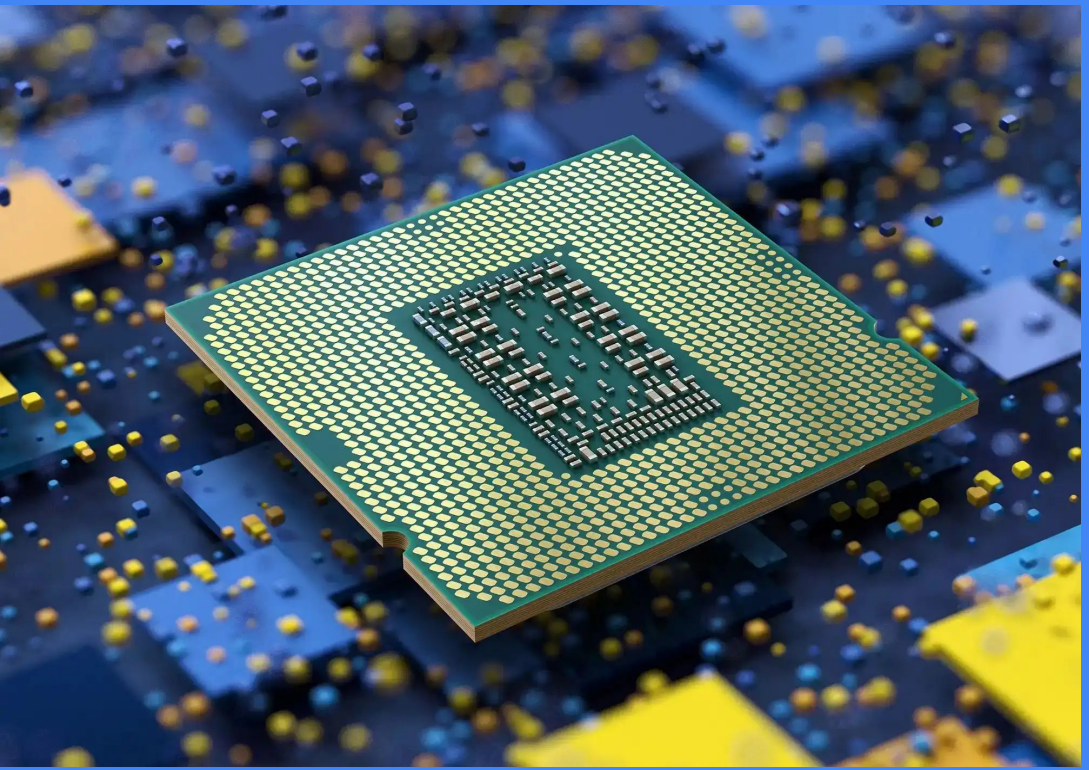
Use BBRv2 congestion control

Resize TCP max send and receive buffer size to twice of target BDP

(sysctl net.ipv4.tcp_[rw]mem)

Disable tcp_peer_metrics that cause early slow-start exit

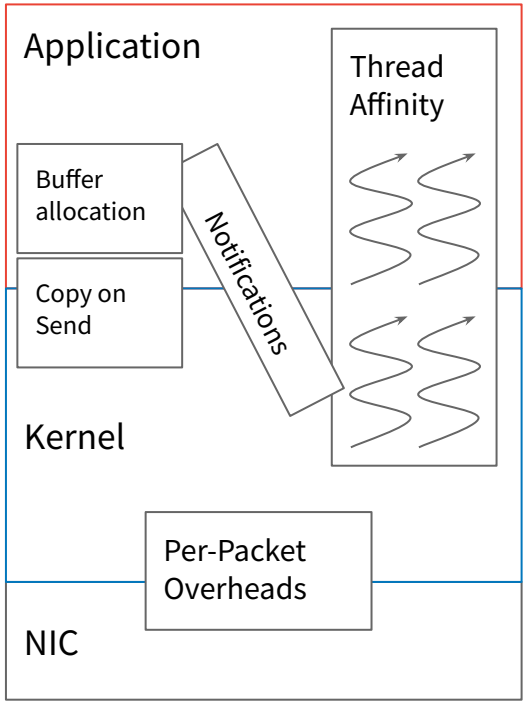
(sysctl net.ipv4.tcp_metrics_save=0)



Optimize TCP
CPU & memory efficiency

TCP bottlenecks in the host

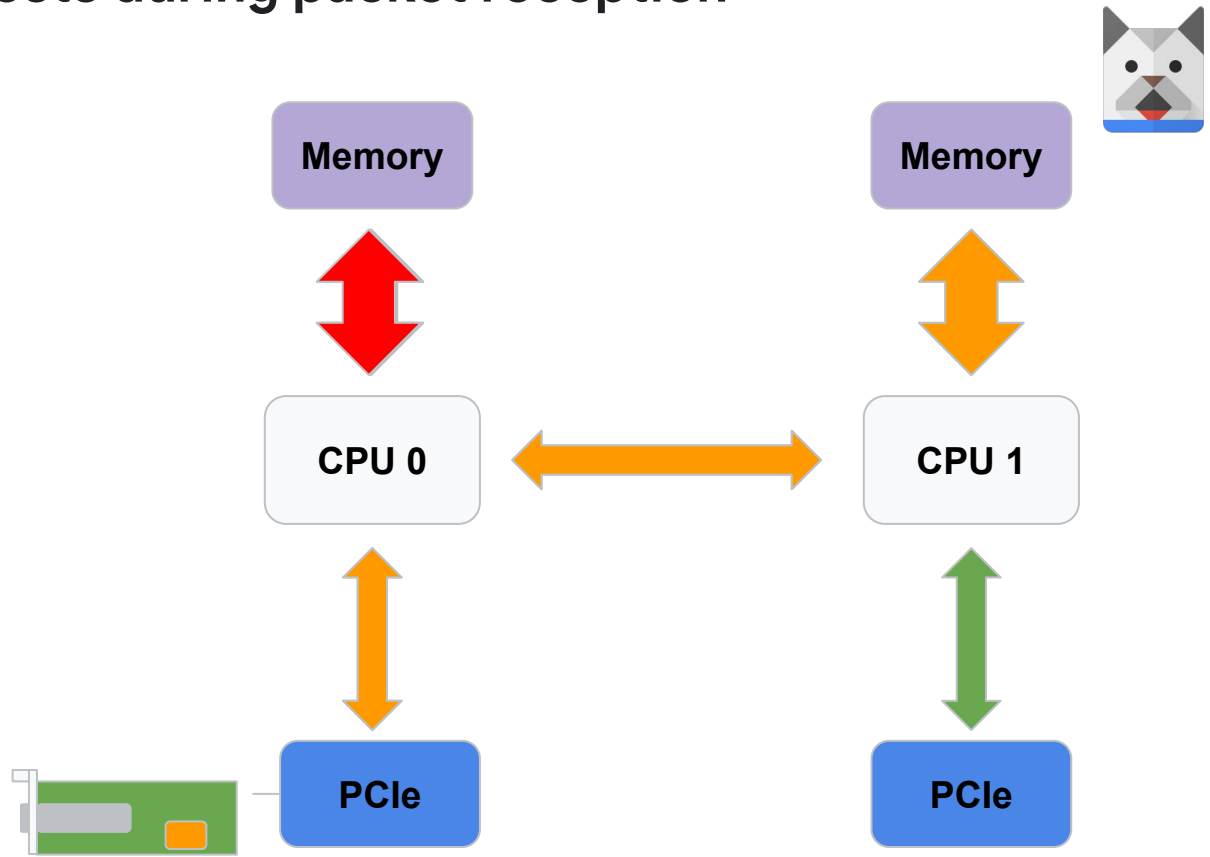
- Copies upon send and receive are expensive (**CPU** & **Memory bandwidth**)
- Userspace and kernel processes handling the same socket can be scheduled on **different CPUs**
- Userspace threads can be **woken up many times** before they can actually do useful work
- System call overheads



Thread Affinity

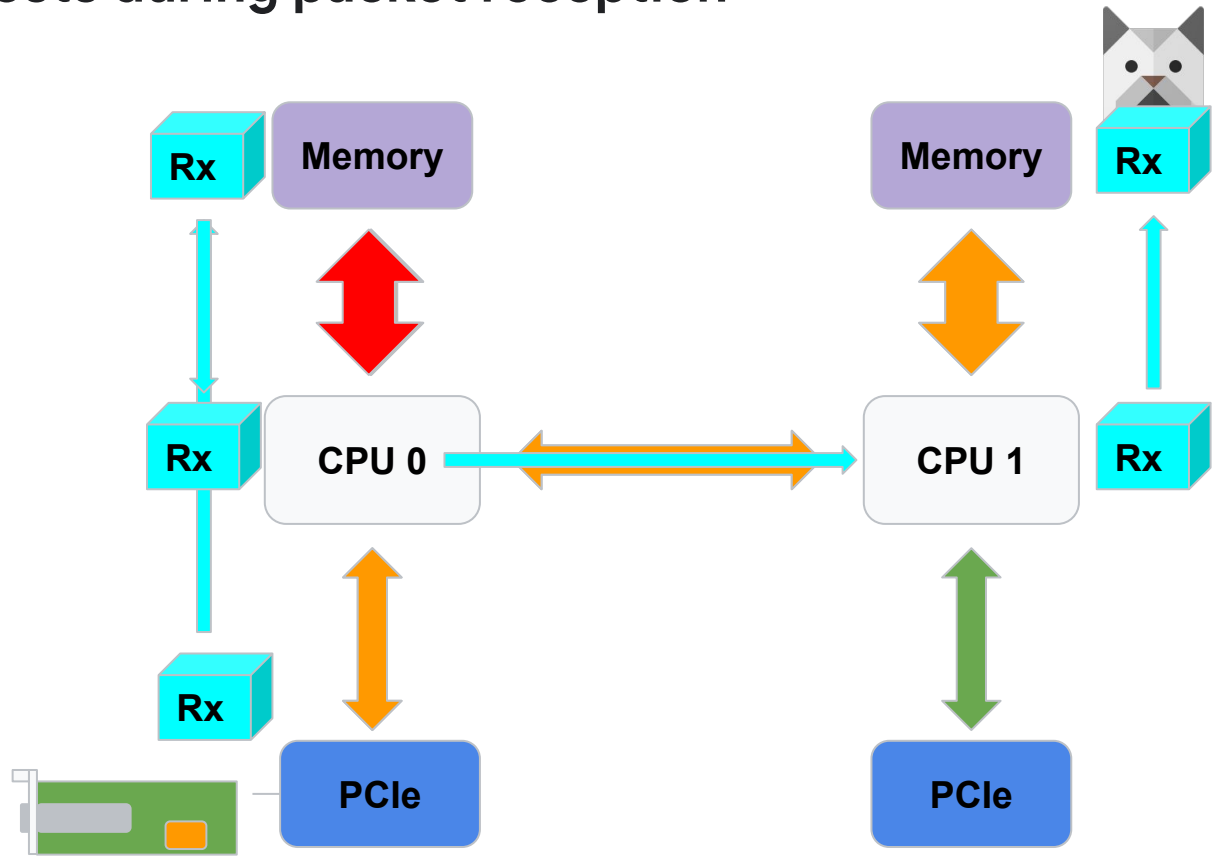
- TCP tries to process packets on the same CPU where the user thread is expected to read/write the data
  - See receive flow steering as an example
- The heuristics are very simple:
  - Set the core ID of the socket when recvmsg and sendmsg are called
  - TCP used to set the core ID on poll, which we removed
- These heuristics won't work without orchestration:
  - Scheduler can move user-space threads around, the previous user space thread may get blocked, pinning doesn't work for all processes, ...
- Orchestrating userspace and kernel affinities we observe 10%+ gains in efficiency and latency:
  - Try to process TCP events on the same core both in userspace and the kernel

Example: costs during packet reception



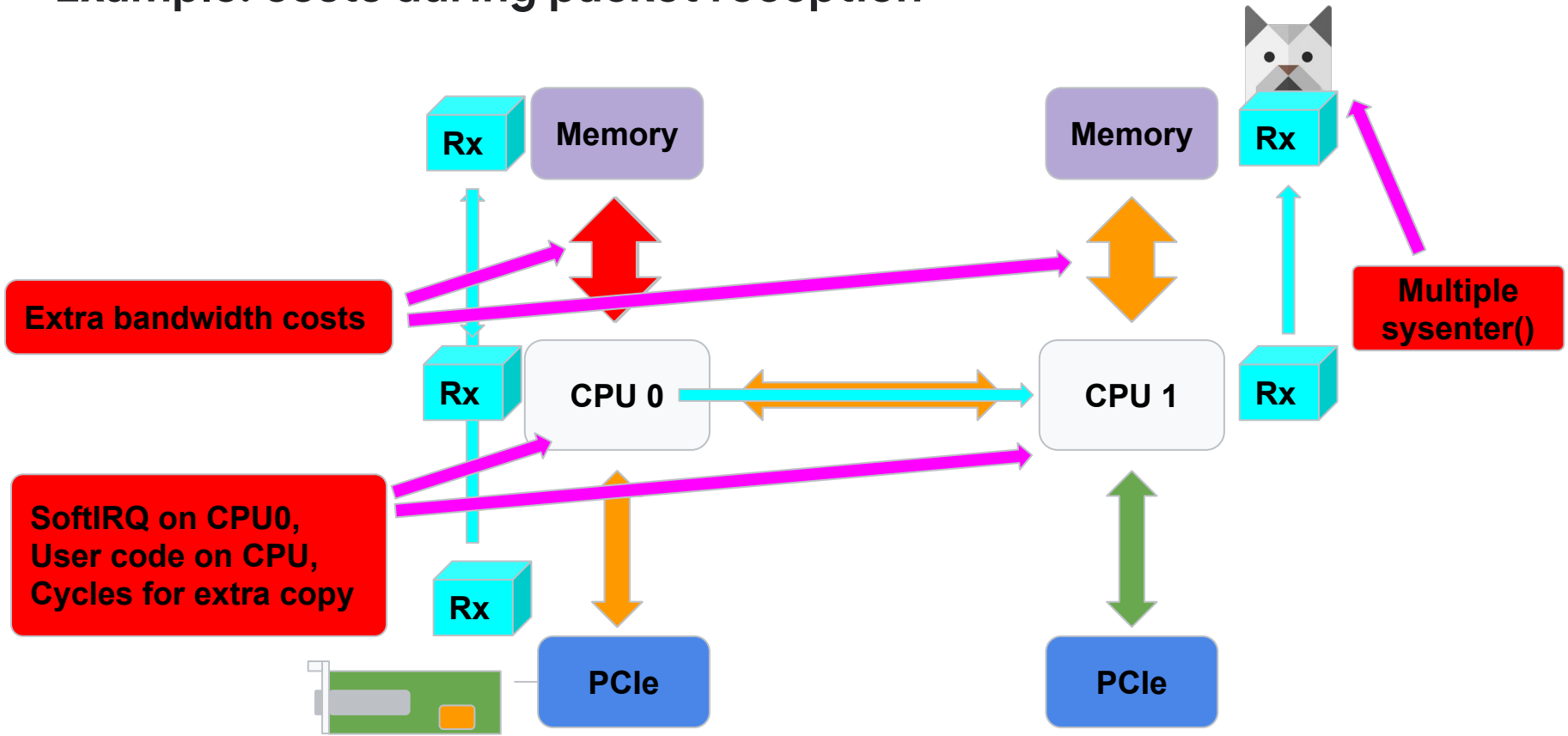
(Arrow thickness correlates with bandwidth)

Example: costs during packet reception

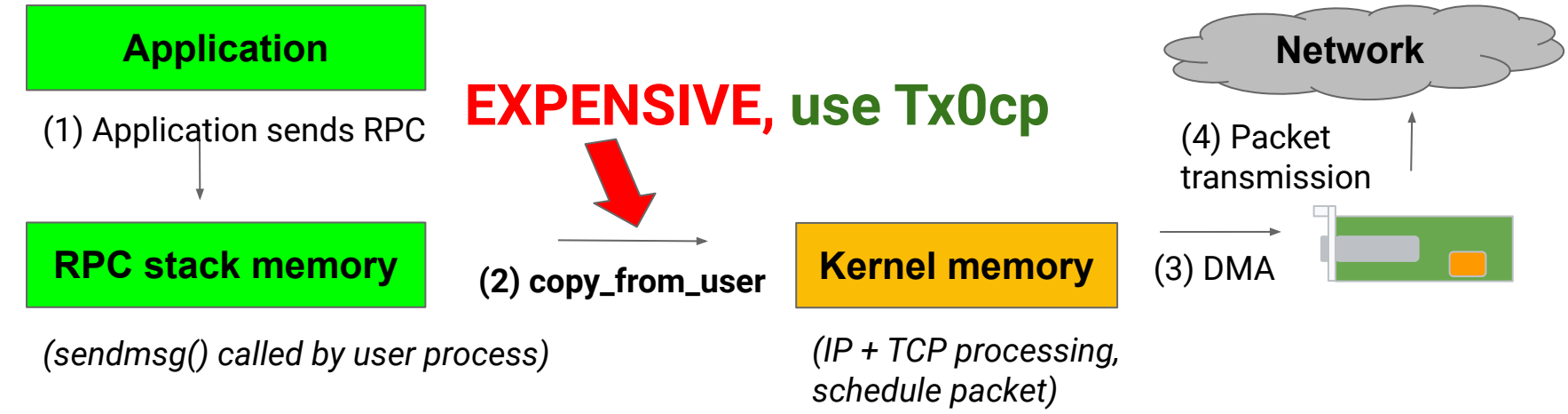


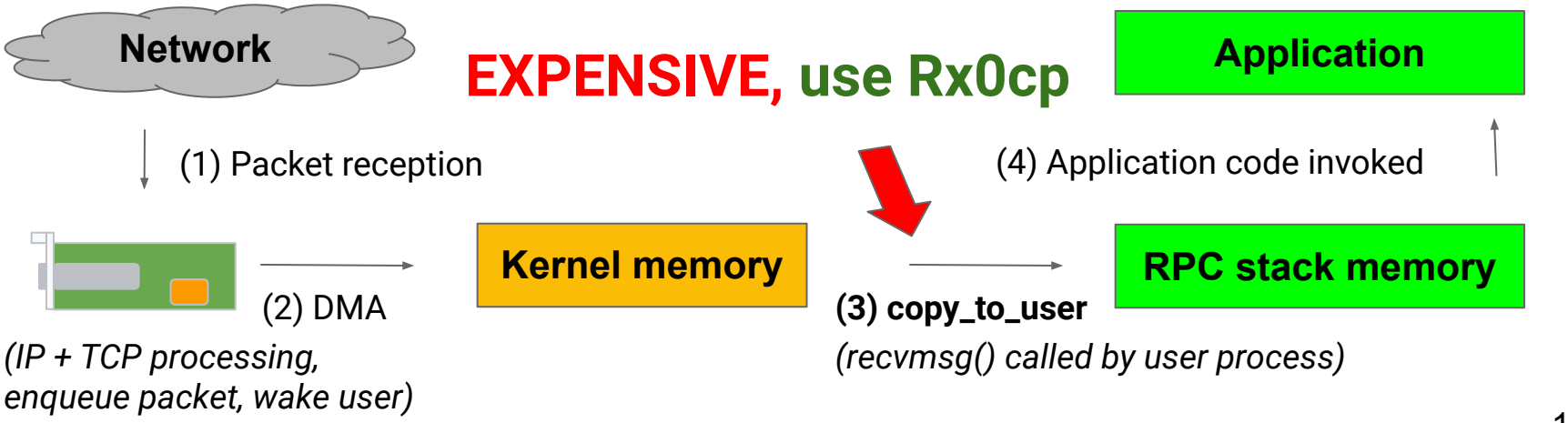
(Arrow thickness correlates with bandwidth)

Example: costs during packet reception



(Arrow thickness correlates with bandwidth)


Application
(1) Application sends RPC
RPC stack memory
(sendmsg() called by user process)
EXPENSIVE, use Tx0cp
(2) copy_from_user
Kernel memory
(IP + TCP processing, schedule packet)
(3) DMA
Network
(4) Packet transmission


Network
(1) Packet reception
(2) DMA
(IP + TCP processing, enqueue packet, wake user)
Kernel memory
EXPENSIVE, use Rx0cp
(3) copy_to_user
(recvmsg() called by user process)
Application
(4) Application code invoked
RPC stack memory

Application-sensible TCP telemetry

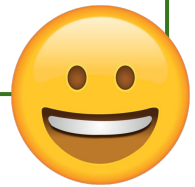
What's the network performance of my RPCs?

Hard to infer from existing TCP connection stats (TCP_INFO)

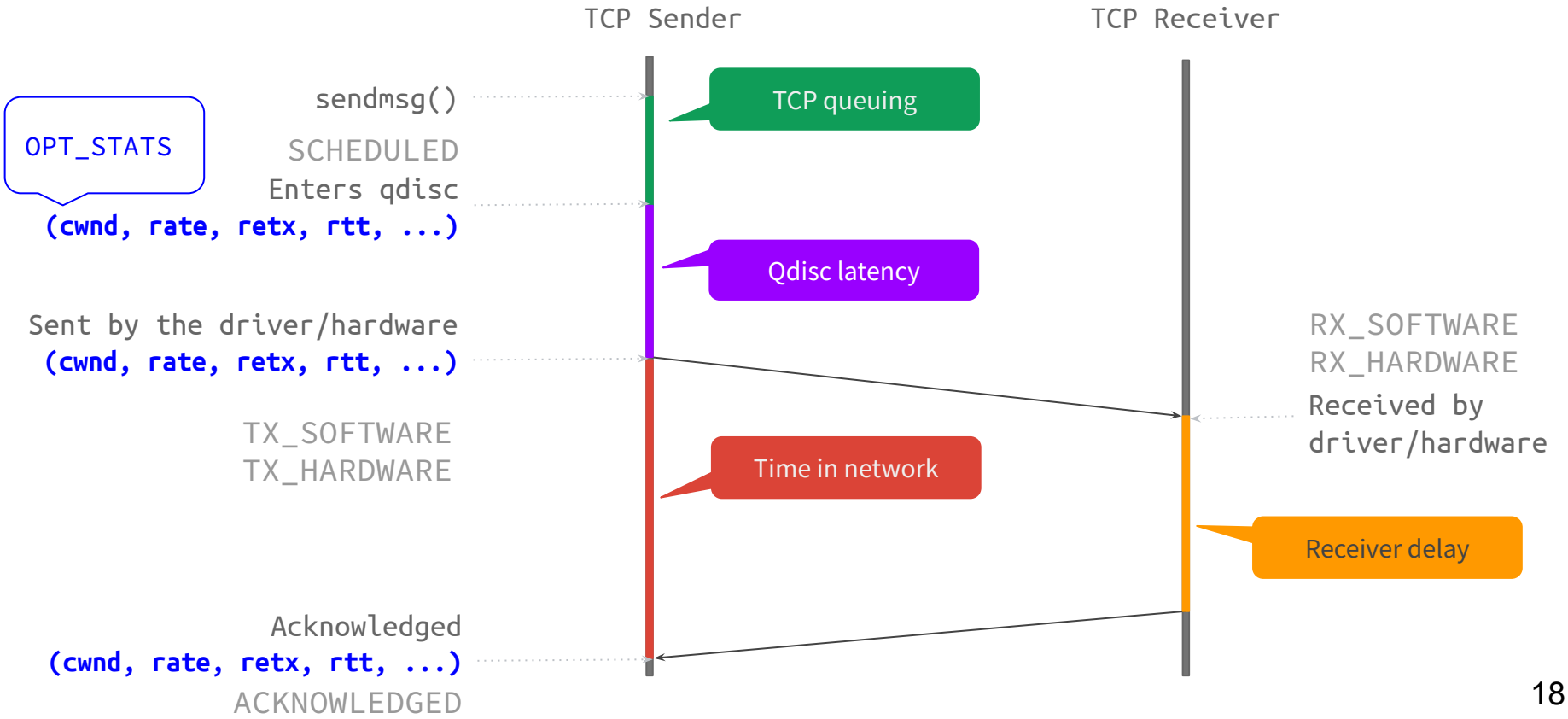
The TCP flow of the response has an average RTT of 10 ms, 3% loss, average congestion window of 31 pkt, ...

New approach: stats associated with application message (SO_TIMESTAMPING)

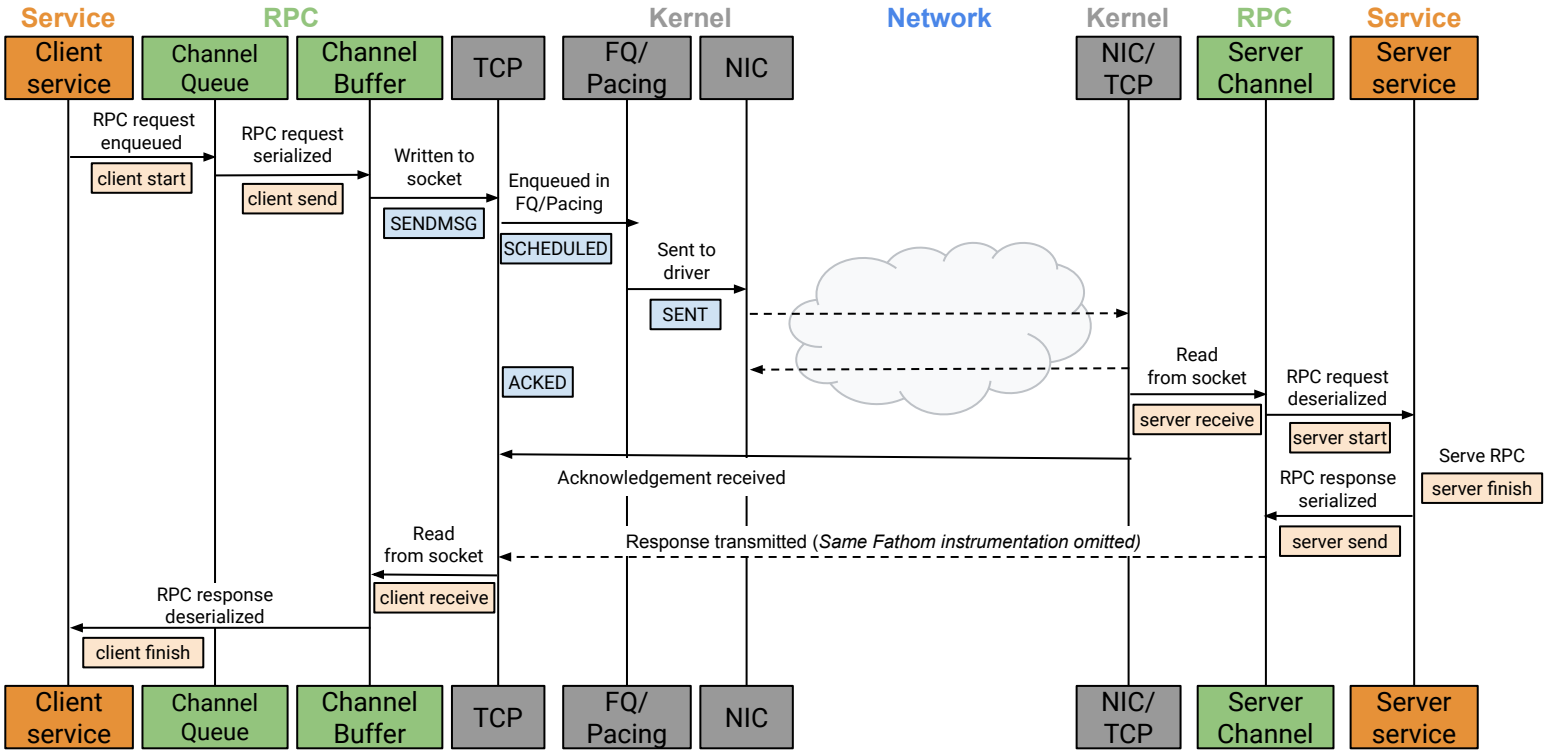
Your RPC response took 45ms to deliver by TCP with 2% of losses. 30ms out of 45ms was spent on the actual network.



Major events recorded for TCP via SO_TIMESTAMPING



Application sensible RPC & TCP telemetry



Many more TCP optimizations

- TCP memory isolation and performance
  - “TCP memory isolation on multi-tenant servers” talk in LPC Referred Track on Tuesday at 3:45pm
- TCP tx zerocopy and thread affinity (IETF 102 tcpm, 2018)
- BIG TCP: bigger TSO/GRO for +200Gbps networks (netdev 0x15, 2021)
- TCP BBR.swift: a delay based DCN-optimized extension of BBRv2 (IETF 109 iccrg, 2020)
- TCP silent close: FIN flood upon server restart could be a bad move (IETF 112 tcpm 2021)
- Multi-path TCP
  - Tuesday talk at 5:30pm
- Reduce TCP default min_RTO of 200ms (Work in progress)
- TCP microsecond based timestamp options (ask Eric Dumazet)
- And many more ...

Conclusion

- Linux TCP can be optimized substantially for datacenter networks
  - Better congestion control, pacing & mixing, dynamic flowlet
  - Network, CPU, memory performance are all important
- Its performance depends vastly on *how* applications use TCP
  - There is no way to guarantee affinity without user-space orchestration
  - Guide optimizations via application-sensible TCP telemetry
- Linux TCP is highly extensible to meet future challenges
  - ebpf, io_uring, congestion control, MPTCP
  - *Must* care complexity and prevent death of a thousand paper cuts

Sample challenges

- Application data-unit (ADU) awareness
  - E.g., Congestion control and EPOLLINs should be frame-aware.
- Process small ADUs to/from many sockets at once
  - Scalable event handling interface
- Easier protocol extension to evolve
  - New IETF TCP option number takes years