Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions

High Performance Data Center Communication with FlexNIC

Thomas Anderson
Warren Francis and Wilma Kolm Bradley Chair
Paul G. Allen School of Computer Science and Engineering
University of Washington

Data Center Application Trends

- **Small, frequent remote procedure calls**
  - Median RPC ~ 300 bytes across entire Google DC [Montazeri, SIGCOMM 18]
  - TCP semantics: reliable, in-order, congestion/flow control
  - Many to many

- Key-value store, database, distributed analytics …
- Pushing performance limits
  - More requests, larger data sets, real-time response, …
- Scale up to 1000s of machines
  - Need predictable tail latency behavior over multiplexed resources
- With NVM persistence now almost entirely a networking issue

…but software packet processing is too slow

- Recv+send TCP stack processing time (2.2 GHz)
  - Linux: 3.5μs
  - Kernel bypass: ~1μs

- **Single core performance has stalled**
- Parallelize? Assuming 1μs over 100Gb/s, ignoring Amdahl's Law:
  - 64B packets => 200 cores
  - 1KB packets => 14 cores

What About?

- Kernel bypass? (ex: MTCP)
  - Kernel overhead only part of the problem
  - No policy enforcement
- SmartNICs/NIC CPU array? (Cavium, Netronome, …)
  - Complex assignment of flows to CPU array
  - Limited per-flow performance
  - Relatively expensive
- RDMA?
  - RDMA API fine for some apps, but message passing is a better fit for small RPCs
  - Hardware bundles (poorly designed) flow/congestion control with API
- TCP Offload Engine?
  - Need protocol agility

Hardware Assist, OS Feature Set

- Multi-tenant policy compliance
  - VM/container security and access control
  - Shared network resource management (flow and congestion control)
- Protocol agility (across lifetime of the hardware)
  - API agnostic: both RDMA and message passing
  - Reconfigurable protocols vs. fixed function hardware
- Connection scalability: 100K+ active flows/server
- CPU efficiency for common case packet handling
  - From NIC through the application and back
- Performance predictability, esp tail latency with many flows
- Cost-efficient hardware: FPGA or micro-programmed VLSI

Hardware Assist Possible at Several Layers

- Virtual machine layer: Sambhrama
  - Deliver packets directly to the guest OS
  - With VM policy enforcement
- **Container OS layer: TCP packet handling**
  - Deliver packets directly to the application
  - With policy enforcement, flow/congestion control, …
- Application-specific processing: Simon Peter
- **Network switches: congestion and SLA management**

FlexNIC

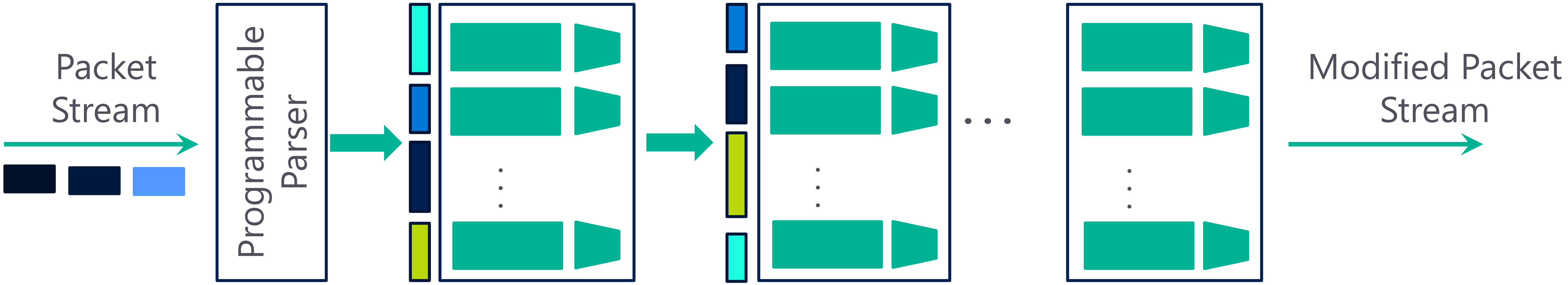
Approx Fair Queueing

Overarching Lesson

Common case packet handling is systolic (can be pipelined in hardware)

On both NICs and switches

FlexNIC: Reconfigurable Multi-stage Pipelines



- Reconfigurable packet processing pipelines
  - Protocol agnostic
  - Tbps implementations for a single pipeline (Barefoot)
  - Predictable performance
- Stages execute parallel

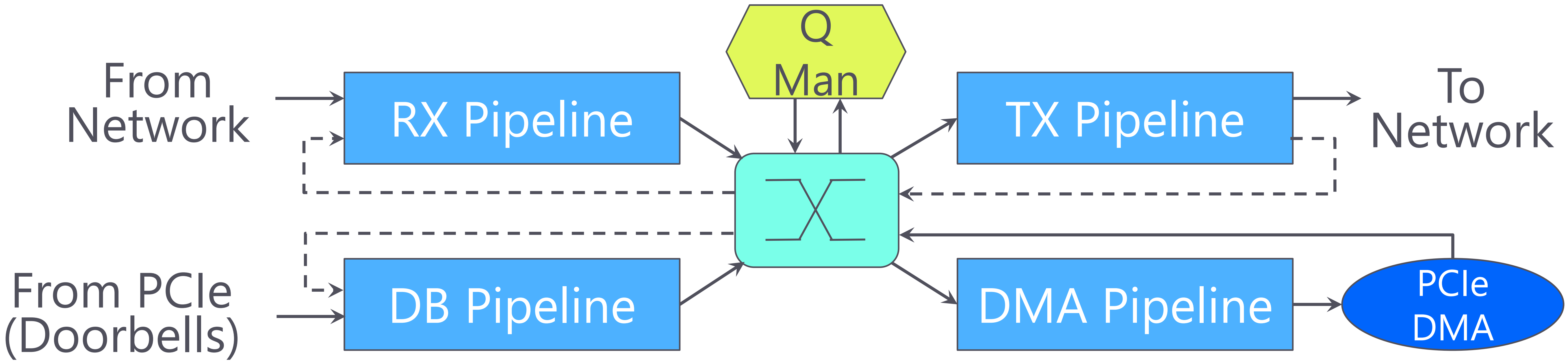
Match+Action Programs

Match:
IF udp.port == kvs
Action:
core = HASH(kvs.key) % 2
DMA hash, kvs **TO** Cores[core]

Supports:
- Steer packets
- Initiate DMA
- Trigger reply packet
- Modify/replicate packets
- Modest per-flow state

Does not support
- Loops
- Complex arithmetic
- Arbitrary state
- Arbitrary # of stages

FlexNIC Hardware Model



- Transform packets for efficient processing in SW
- DMA directly into and out of application data structures
- Send acknowledgements on NIC
- Queue manager implements rate limits
- Improve locality by steering to cores based on app criteria

Kernel

- Open/close connections

Per packet

- Socket API, locking
- IP routing, ARP
- Firewalling, traffic shaping
- Generate data segments
- Congestion control
- Flow control
- Process & send ACKs
- Re-transmission timeouts

Complex connection state spread over multiple data structures, multiple queues, pointer chasing, …

Application

- Socket API, locking

Fast Path: FlexNIC

Per packet: constant time operations

- Generate data segments
- Apply rate-limit
- Congestion statistics
- Flow control
- Process & send ACKs

Minimal Connection State: 100 bytes

Slow Path: Kernel

- Open/close connections
- IP routing, ARP
- Firewalling, traffic shaping
- Compute rate
- Re-transmission timeouts

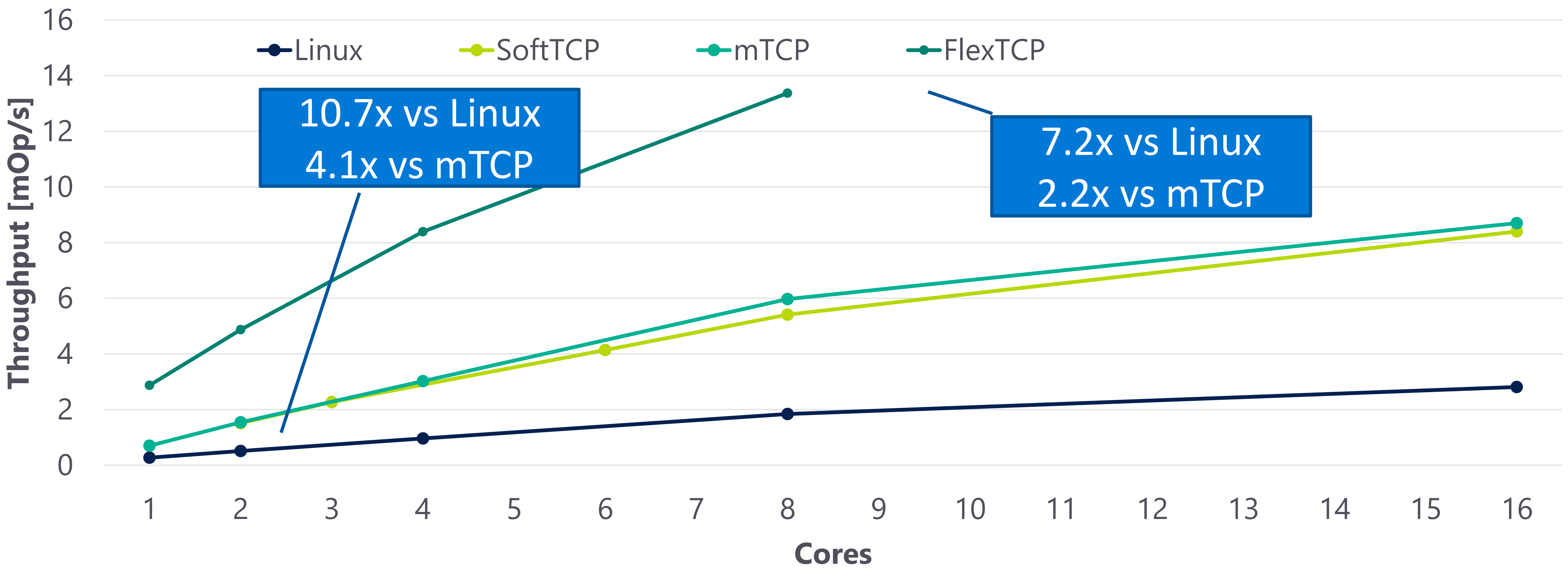
Periodic Congestion Control

- Linux TCP: per-packet congestion window calculation
  - Ack clocking triggers packet queuing for transmission
  - Liable to starvation as # of flows increases

- FlexTCP: per-RTT rate limit
  - FlexNIC: enforce rate-limit, collect CC statistics
  - Kernel software: Fetch CC statistics, update rate-limit
  - Congestion statistics: # ACKs, # ECN marks, # drops, RTT estimation

- Not specific to congestion algorithm
  - Implemented DCTCP, TIMELY, and Reno

FlexTCP Performance



- Latency: 7.8x better vs Linux
- FlexNIC per-flow isolation vs. Linux per-flow starvation

Fair Queueing: in-network enforcement

Enforce fair allocation and isolation at switches

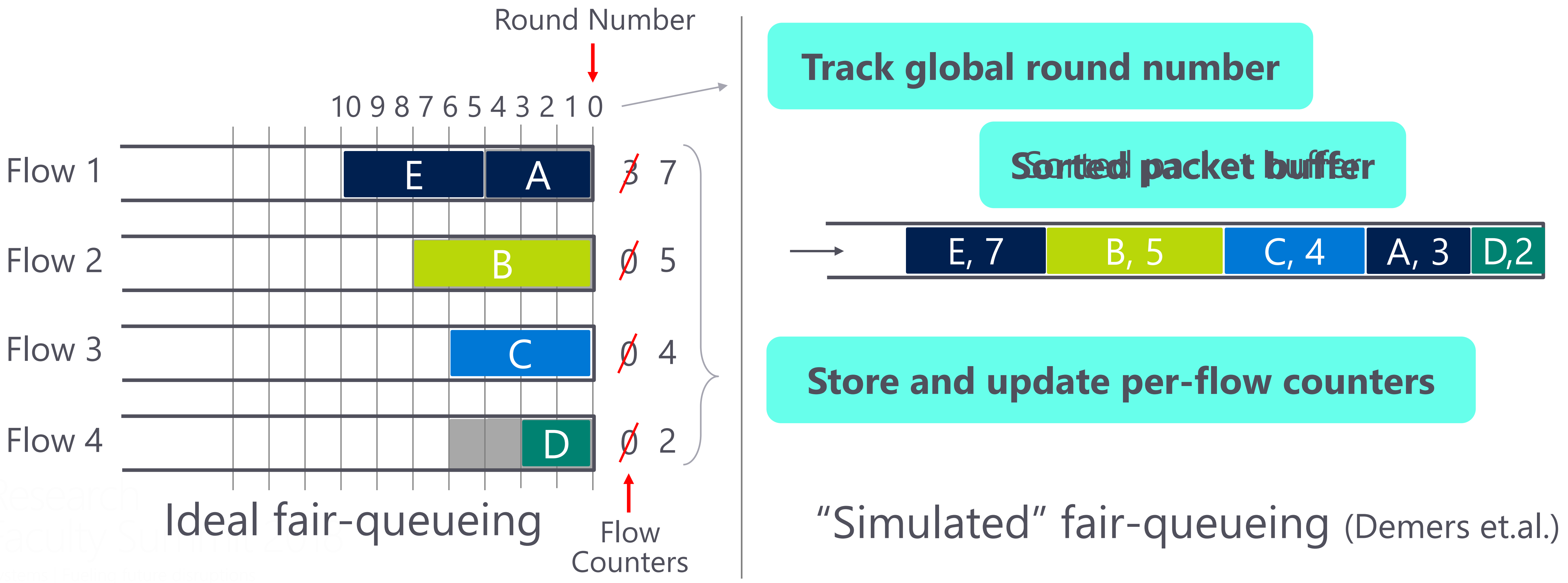
- Provide an illusion that every flow has its own queue
- Proven to have perfect isolation and fairness

+ Simplifies congestion control at the end-host
+ Protects against misbehaving traffic
+ Enables bounded delay guarantees

However, challenging to realize in high-speed switches.

Fair Queueing without per-flow queues

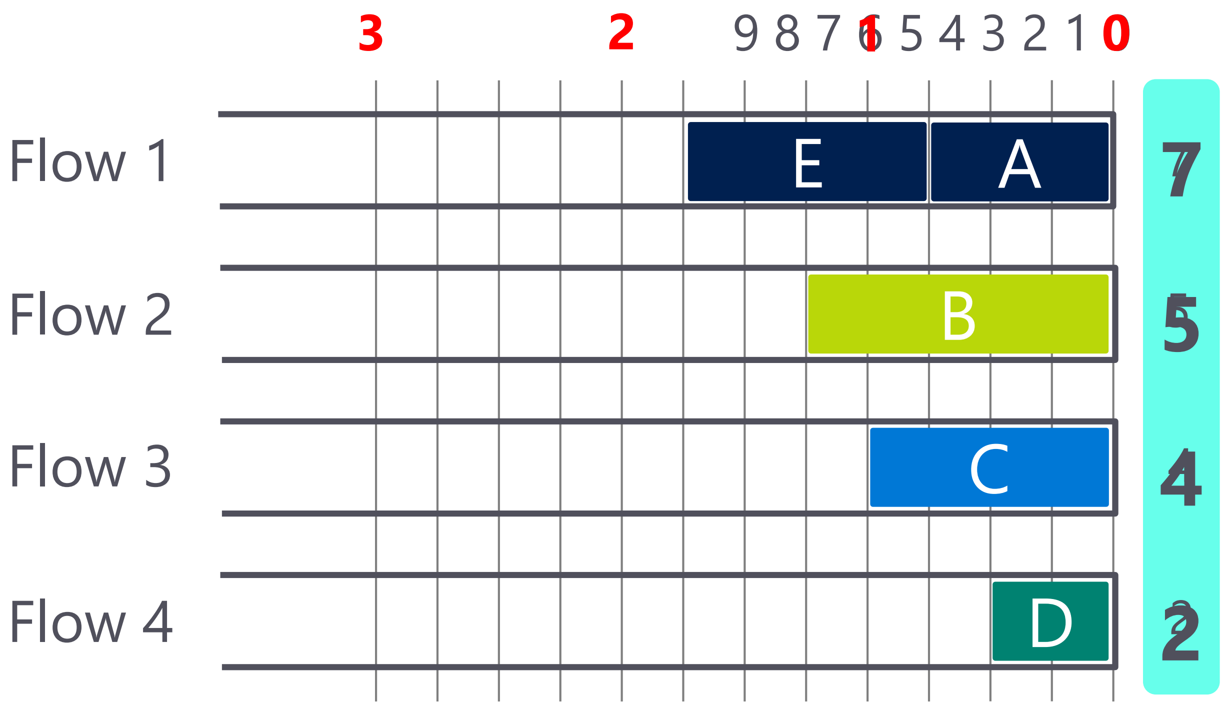
- *Simulates* an ideal round-robin scheme where each active flow transmits a single bit of data every round.



Our approach: Approximate Fair Queueing

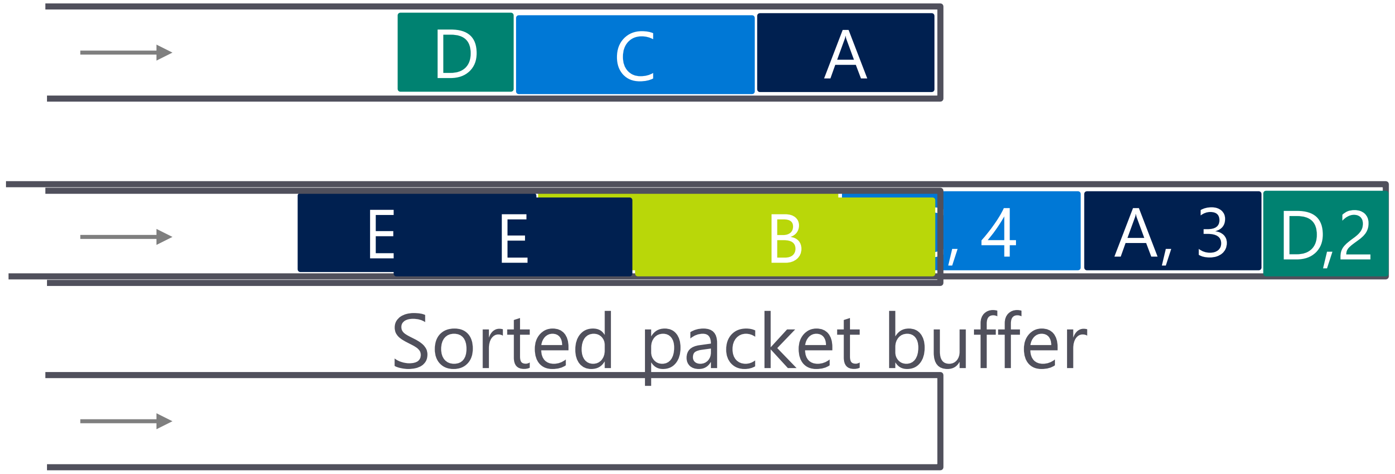
Simulate a bit-by-bit round robin scheme with key approximations

Coarse round numbers



Ideal fair-queueing

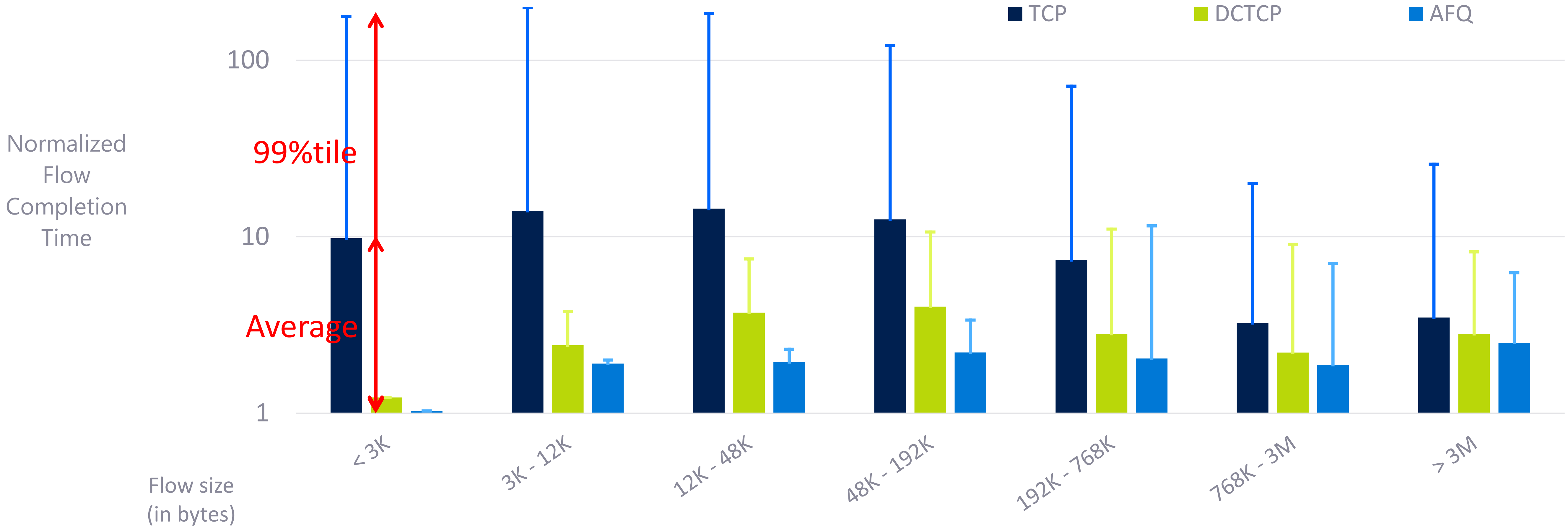
Limited # of FIFO queues with rotating priorities to approximate a sorted buffer



Store approximate per-flow counters using a variation of the count-min sketch

queueing

Testbed Results



- Compared to TCP, 4x better average FCT, 10x better tail latency
- Compared to DCTCP, 2x better average FCT, 4x better tail latency

Summary

- FlexNIC
  - Configurable, efficient, policy-compliant NIC packet handling
  - For VM, container, application
  - Key idea: common case behavior as match-action, kernel for exception handling

- Approximate fair queueing with switch match-action tables
  - Configurable, efficient, policy-compliant switch packet handling
  - Fair queueing provides performance isolation, network SLAs, QoS
  - Approximate with rotating priority queues, coarse-grained rounds, approx. per-flow counters

Thank you

Microsoft