Approximating Fair Queueing on Reconfigurable Switches

Naveen Kr. Sharma, Ming Liu, Kishore Atreya, Arvind Krishnamurthy

UNIVERSITY *of* WASHINGTON

CAVIUM

Congestion Control Today

Primarily achieved using end-to-end protocols (TCP, DCTCP, TIMELY, ..)

- End-hosts react to signals from the network.
- Network does not enforce fair sharing or isolation.

\+ Requires minimal network support

\+ Switches can operate at very high speeds

\- End-host must cooperate to achieve fairness

\- Leads to several inefficiencies and poor isolation

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

Analysis and Simulation of a Fair Queueing Algorithm

Alan Demers
Srinivasan Keshav†
Scott Shenker
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Abstract

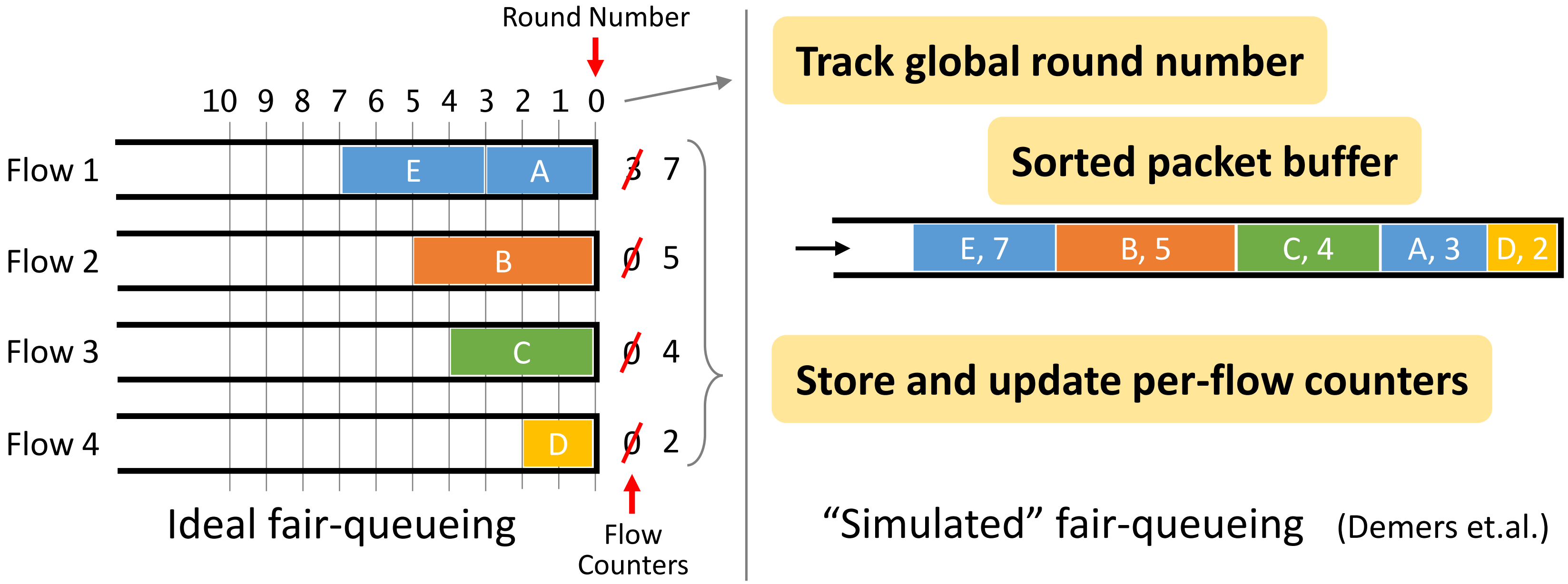
We discuss gateway queueing algorithms and their role in controlling congestion in datagram networks. A fair queueing algorithm, based on an earlier suggestion by Nagle, is proposed. Analysis and simulations are used to compare this algorithm to

often ignored, makes queueing algorithms a crucial component in effective congestion control.

Queueing algorithms can be thought of as allocating three nearly independent quantities: bandwidth (*which* packets get *transmitted*), promptness (*when* do those packets get *transmitted*), and buffer space (*which* packets are *discarded* by the gateway)

Fair Queueing without per-flow queues

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

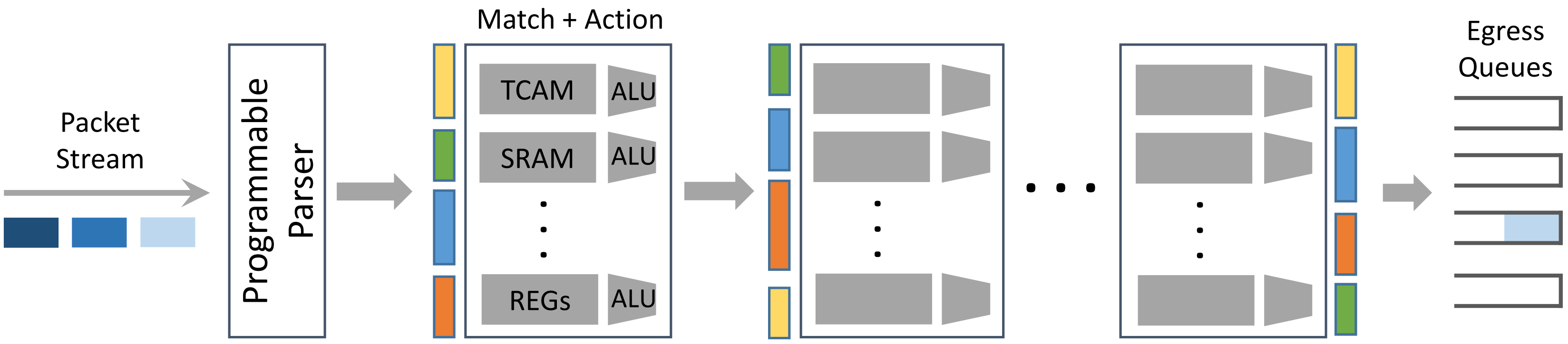


Rest of the talk

- Background: Reconfigurable Switches
- Our approach: Approximate Fair Queueing
- Optimized End-host Flow Control Protocol
- Prototype Implementation
- Evaluation

Reconfigurable Switches

- New class of programmable switches that allow customizable data-plane



- TCAM, SRAM for table lookups or matches
- ALUs for modifying headers and registers
- Stateful memory for counter and meters

- `port = lookup(eth.dst_mac)`
- `ipv4.ttl = ipv4.ttl - 1`
- `counter[ipv4.dst_port]++`

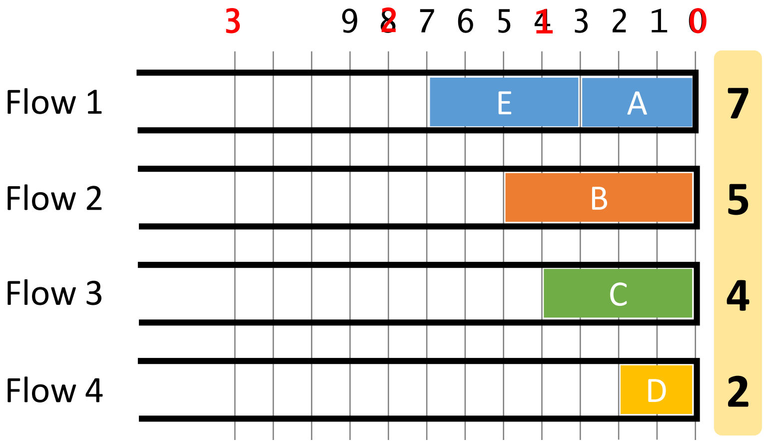
Realizing Fair Queueing on Reconfigurable Switches

1. Maintain a sorted packet buffer
   - *Requirement*: O(logN) insertion complexity
   - *Constraint*: Limited operations per packet
2. Store per-flow counters
   - *Requirement*: Per-flow mutable state
   - *Constraint*: Limited switch memory
3. Access and modify current round number
   - *Requirement*: Synchronize state across switch modules
   - *Constraint*: Limited cross-module communication

Our approach: Approximate Fair Queueing

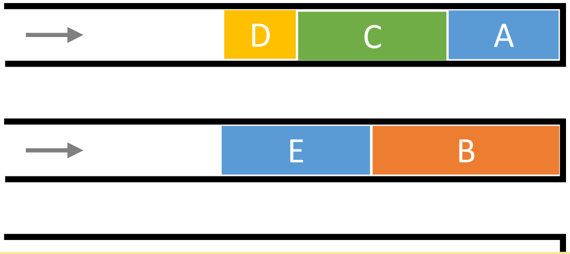
Simulate a bit-by-bit round robin scheme with key approximations

Coarse round numbers



Ideal fair-queueing

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

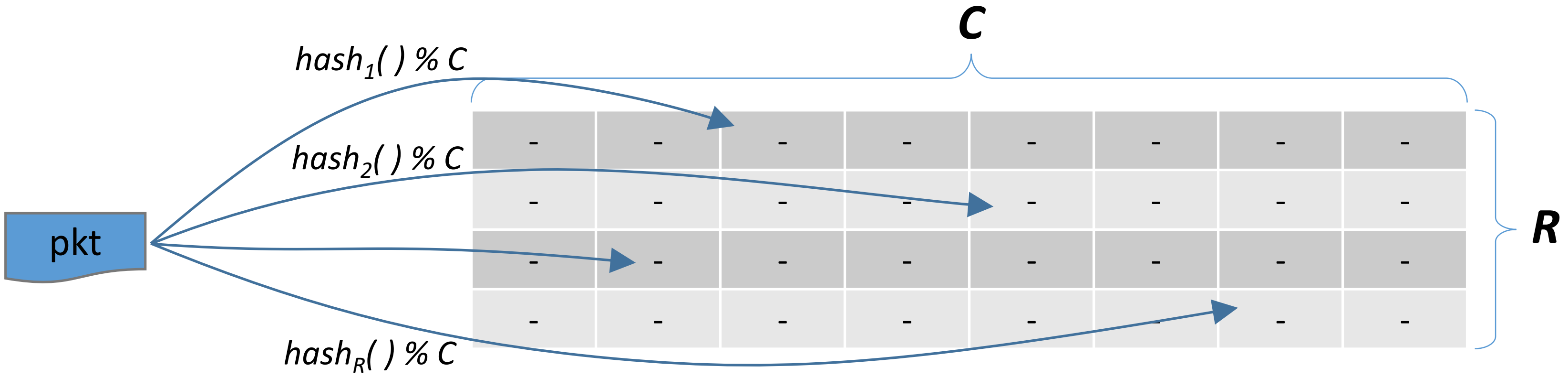


Store approximate per-flow counters using a variation of the count-min sketch

“Simulated” fair-queueing

Storing Approximate Flow Counters

- Variation of count-min sketch to track flow's finish round number



- **update** increments all cells; **read** returns the minimum
- Never under-estimates, has provable space-accuracy trade-off

- Customized to perform combined read-update operation
- Conditional increment upto the new value for better accuracy

Flow 1 size : 1000

Flow 2 size : 500

0	0	1000	0	0	500	0	0
0	0	0	0	1000	0	0	0
0	1000	0	500	0	0	0	0
0	0	0	0	0	500	1000	0

```
min (0, 1000, 0, 0) = 0 + 500 = 500
```

Read Counter

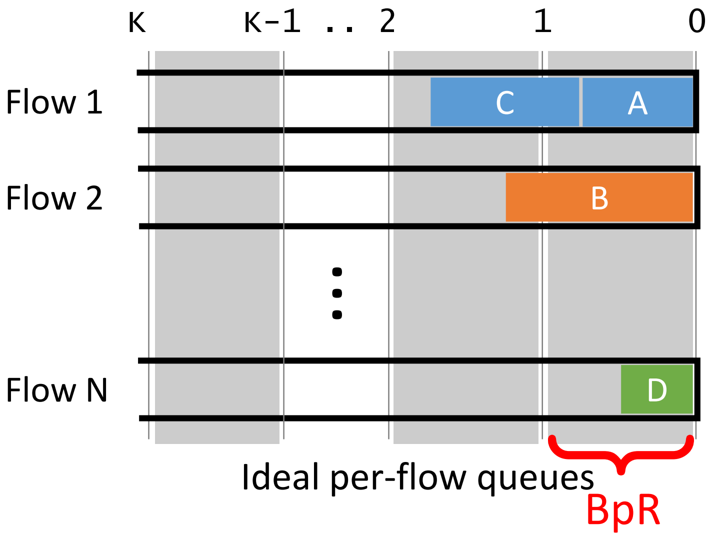
- Find the minimum of all cells
- Bytes sent = minimum + pkt.size

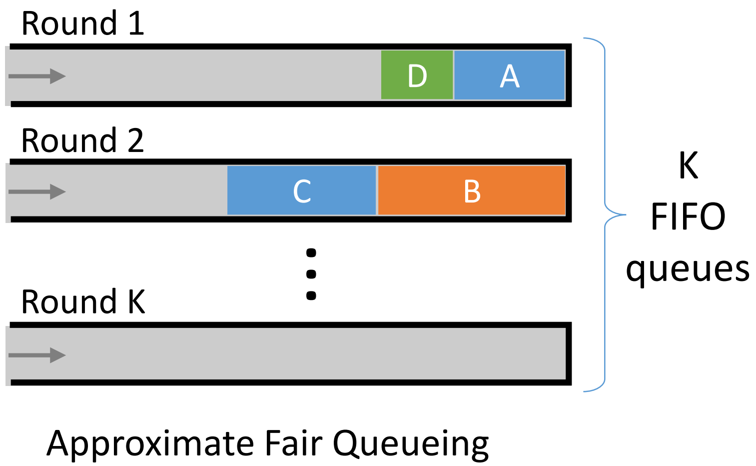
Update Counter

- Increment all cells upto new value
- $cell^{x,y}$ = max ($cell^{x,y}$, new value)

- Implemented in hardware using *predicated* read-write registers

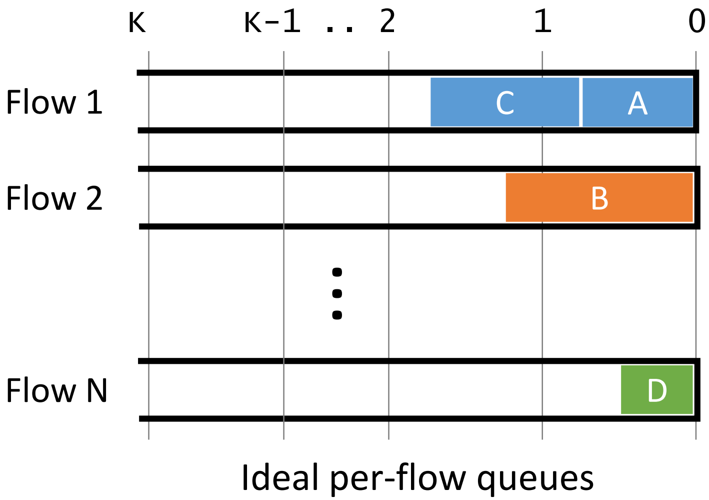
Buffering Packets in Approximate Sorted Order

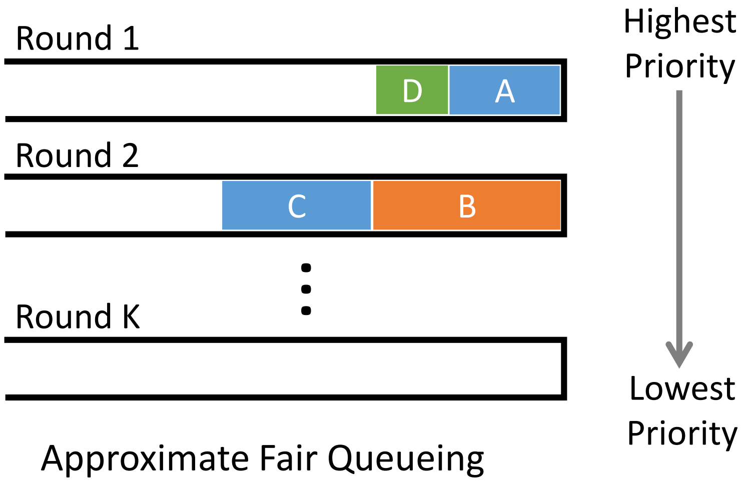




- Coarse rounds: flows transmit a quantum of bytes per round (BpR)
- For each packet, outgoing round number = bytes sent / BpR

Rotating Strict Priority (RSP)





- Drain queue with the lowest round number till it is empty
- Push queue to lowest priority; increment round number by 1

Realizing an RSP Scheduler

RSP can be implemented in hardware

- Identical complexity to a Deficit Round Robin scheduler

RSP can be emulated on current switches

- Switch CPU to periodically change priorities
- Hierarchical priority queues

Avoid explicit round number synchronization by exposing queue metadata

Utilize dynamic buffer sharing to vary size of individual queues

Summary of Techniques

1. Modified count-min sketch
   + Counters for large number of flows in limited memory
   - Collisions cause packets to enqueue in a later round
2. RSP queues to approximate sorted buffer
   + Process packets in fixed number of operations
   - Packets can be reordered within a round
3. Coarse round numbers
   + Updates to shared state are not per-packet anymore
   - Packets can enqueue in an earlier round

Enhancing AFQ with End-host Flow Control

- AFQ can be deployed without modifying end-hosts.
- Adapt the packet-pair algorithm [Keshav, 91] to gain even more benefits.
  - Sender transmits a pair of back-to-back packets.
  - Inter-arrival delay is an estimate of the bottleneck bandwidth.
  - End-hosts send packets at estimated rate.
- Lets us perform fast ramp-up and keep small queue sizes.

Evaluation

- Does AFQ improve overall performance?
- What is the impact of approximations?

This talk

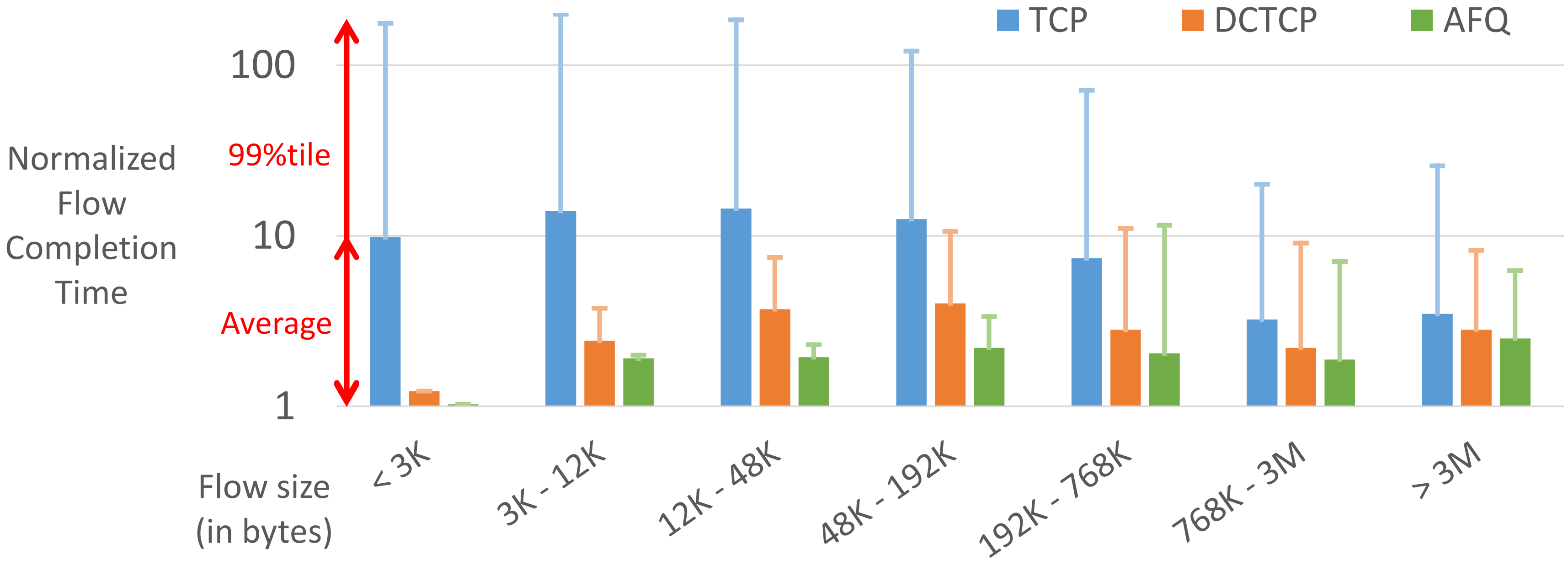
- Can AFQ deal with incast traffic patterns?
- How many FIFO queues are sufficient?
- What size count-min sketch is required?
- How do we set the *BpR* parameter?

In the paper

Prototype Implementation

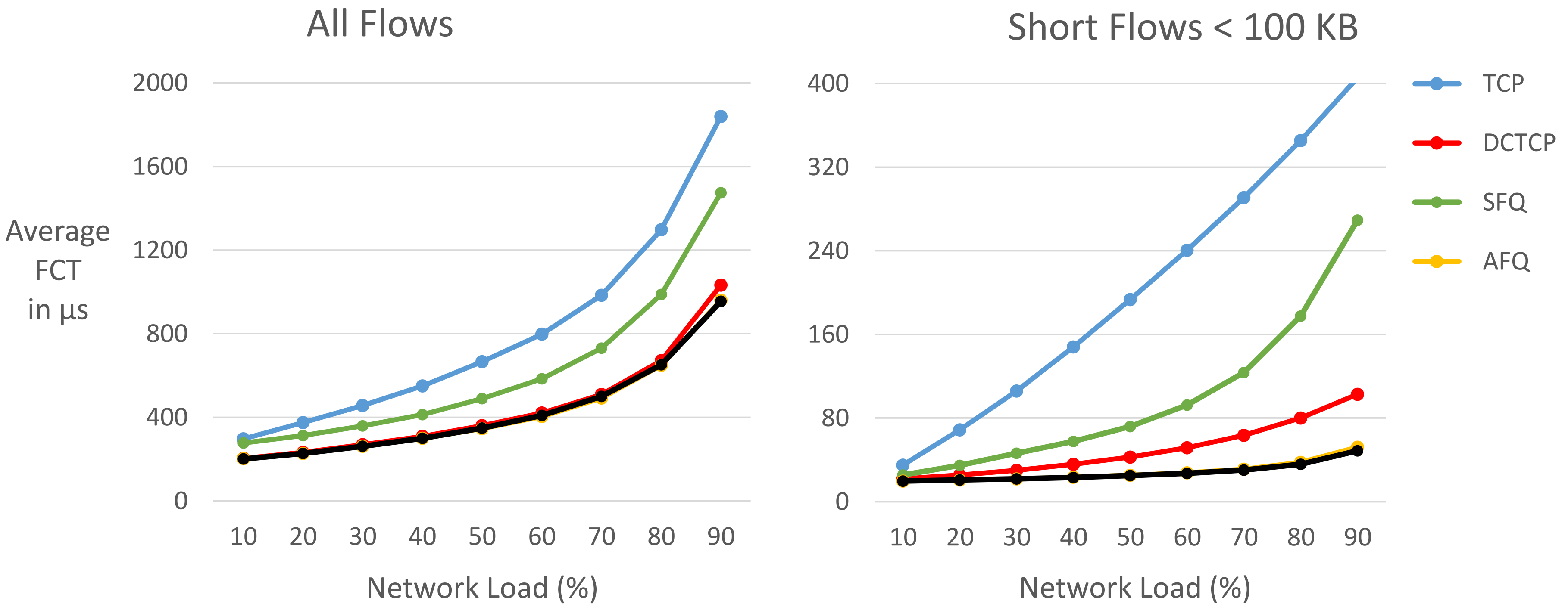
- Built a 4-port AFQ switch atop a Cavium Network Processor.
- Pipelines run on MIPS CPU; packets & counters stored in DRAM.
- Each port runs at 10gbps with 32 FIFO queues and 4x16k sketch.
- Tested on a 2-level fat-tree topology.
- Implemented packet-pair flow control in user space.

Testbed Results



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

Simulation Results: Comparison to Fair Queueing



Other Results

- Accurate approximation achieved using 12 to 16 FIFO queues.
- Less than 10% extra resource overhead on top of switch.p4.
- Significant improvement even with existing end-host protocols.
- Provides ideal fairness during incast traffic patterns.
- Reduces drops and retransmissions by 10x compared to DCTCP.

Summary

- Practical implementation of Fair Queueing at line-rate.
- Use approximation techniques to overcome hardware constraints.
  - Modified sketch to store per-flow counters
  - Leverage limited FIFO queues to approximate sorted buffer
- Approximations are both effective and accurate.
- Leads to 4-8x improvement in flow completion times.