OpenNFP

Accelerating Networked Applications with Flexible Packet Processing

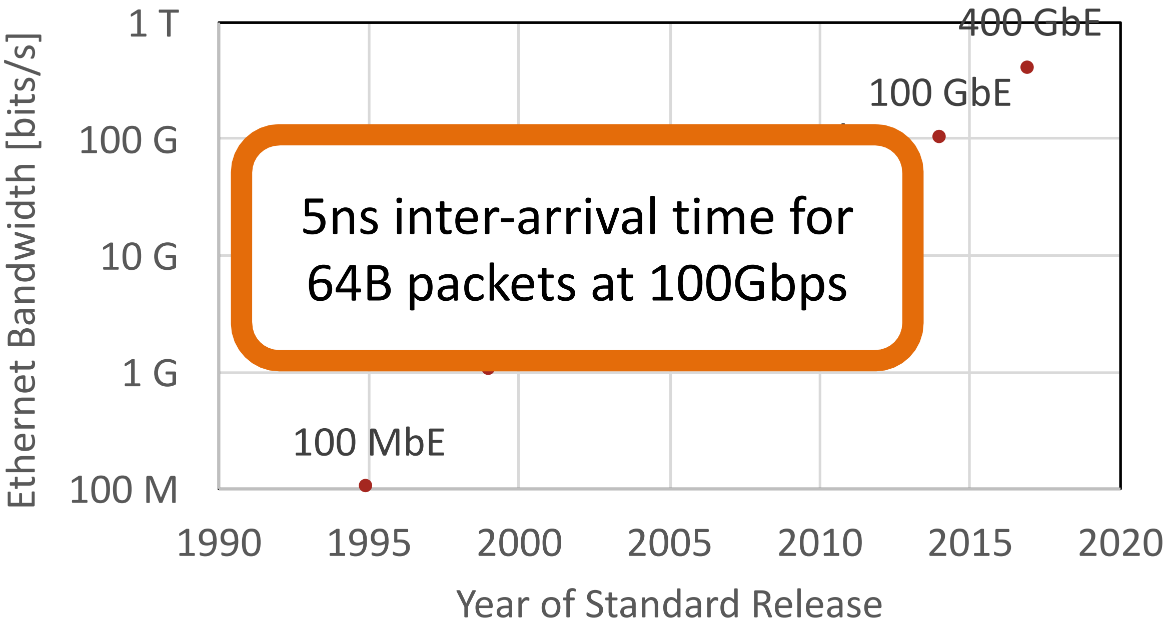
Antoine Kaufmann, Naveen Kr. Sharma, Thomas Anderson, Arvind Krishnamurthy

Timothy Stamler, **Simon Peter**

University of Washington

The University of Texas at Austin

Networks are becoming faster



5ns inter-arrival time for 64B packets at 100Gbps

...but software packet processing is 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, excluding Amdahl's law:

- 64B packets => 200 cores
- 1KB packets => 14 cores

Many cloud apps dominated by packet processing

- Key-value storage, real-time analytics, intrusion detection, file service, ...
- All rely on small messages: latency & throughput equally important

What are the alternatives?

RDMA
- Bypasses server software entirely
- Not well matched to client/server processing (security, two-sided for RPC)

Full application offload to NIC (FPGA, etc.)
- Application now at slower hardware-development speed
- Difficult to change once deployed

Fixed-function offloads (segmentation, checksums, RSS)
- Good start!
- Too rigid for today's complex server & network architecture (next slide)

Flexible function offload to NIC (NFP, FlexNIC, etc.)
- Break down functions (eg., RSS) and provide API for software flexibility

Fixed-function offloads are not well integrated

Wasted CPU cycles

- Packet parsing and validation repeated in software
- Packet formatted for network, not software access
- Multiplexing, filtering repeated in software

Poor cache locality, extra synchronization

- NIC steers packets to cores by connection
- Application locality may not match connection

A more flexible NIC can help

With multi-core, NIC needs to pick destination core

- The “right” core is application specific

NIC is perfectly situated – sees all traffic

- Can scalably preprocess packets according to software needs
- Can scalably forward packets among host CPUs and network

With kernel-bypass, only NIC can enforce OS policy

- Need flexible NIC mechanisms, or go back into kernel

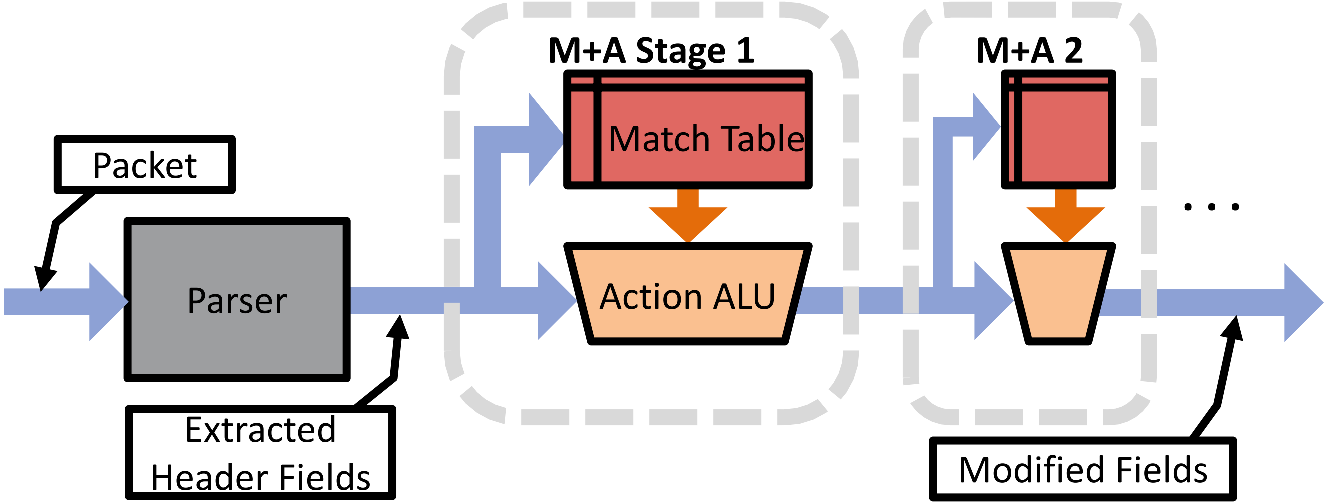
Talk Outline

- Motivation
- FlexNIC model
  - Experience with Agilio-CX as prototyping platform
- Accelerating packet-oriented networking (UDP, DCCP)
  - Key-value store
  - Real-time analytics
  - Network Intrusion Detection
- WiP: Accelerating stream-oriented networking (TCP)

FLEXNIC MODEL

- Implementable at Tbps line rate & low cost

Match+action pipeline:



Match+Action Programs

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

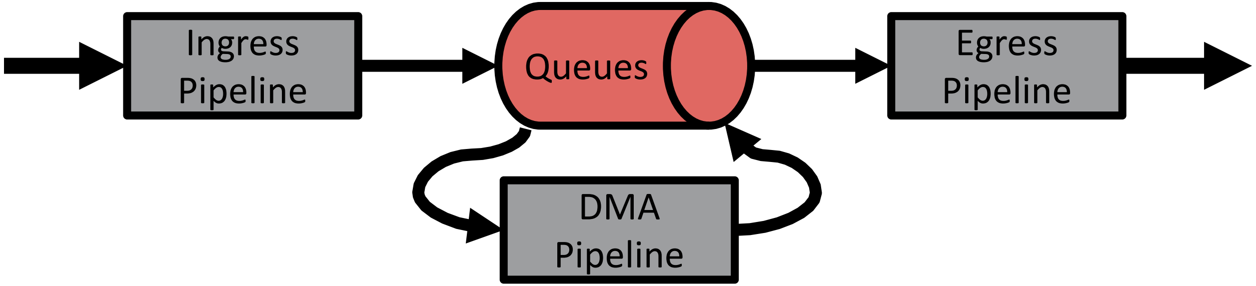
Supports:

Steer packet
Calculate hash/Xsum
Initiate DMA operations
Trigger reply packet
Modify packets

Does not support:

Loops
Complex calculations
Keeping large state

FlexNIC: M+A for NICs



Efficient application level processing in the NIC

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

Netronome Agilio-CX

We use Agilio-CX to prototype FlexNIC

- Implement M&A programs in P4
- Run on NIC

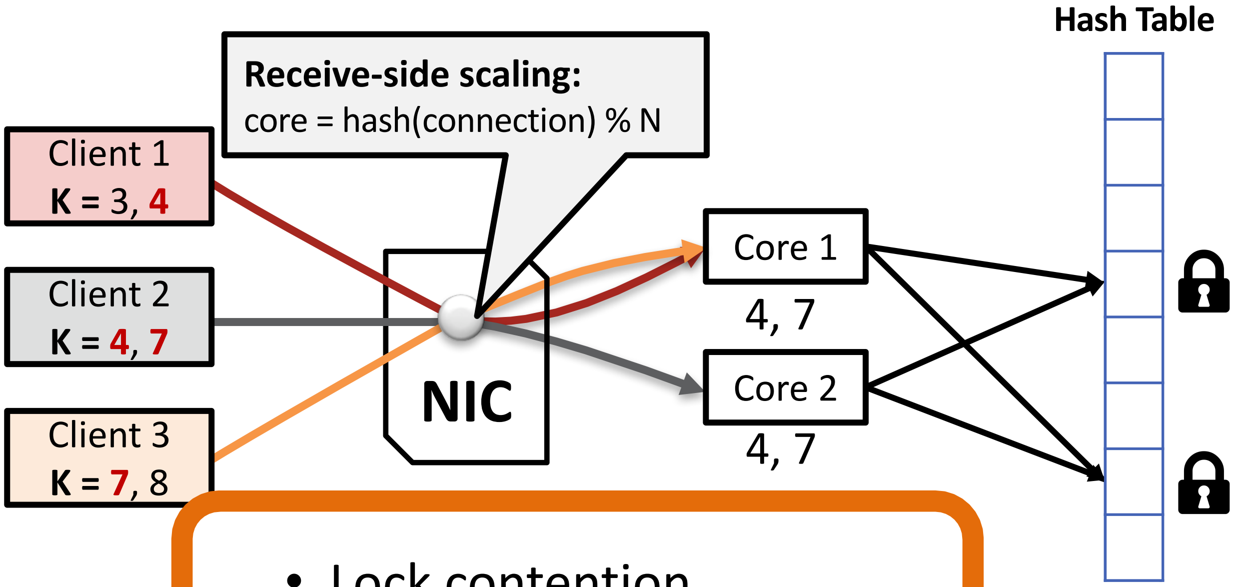
Our experience with Agilio-CX:

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

OpenNFP

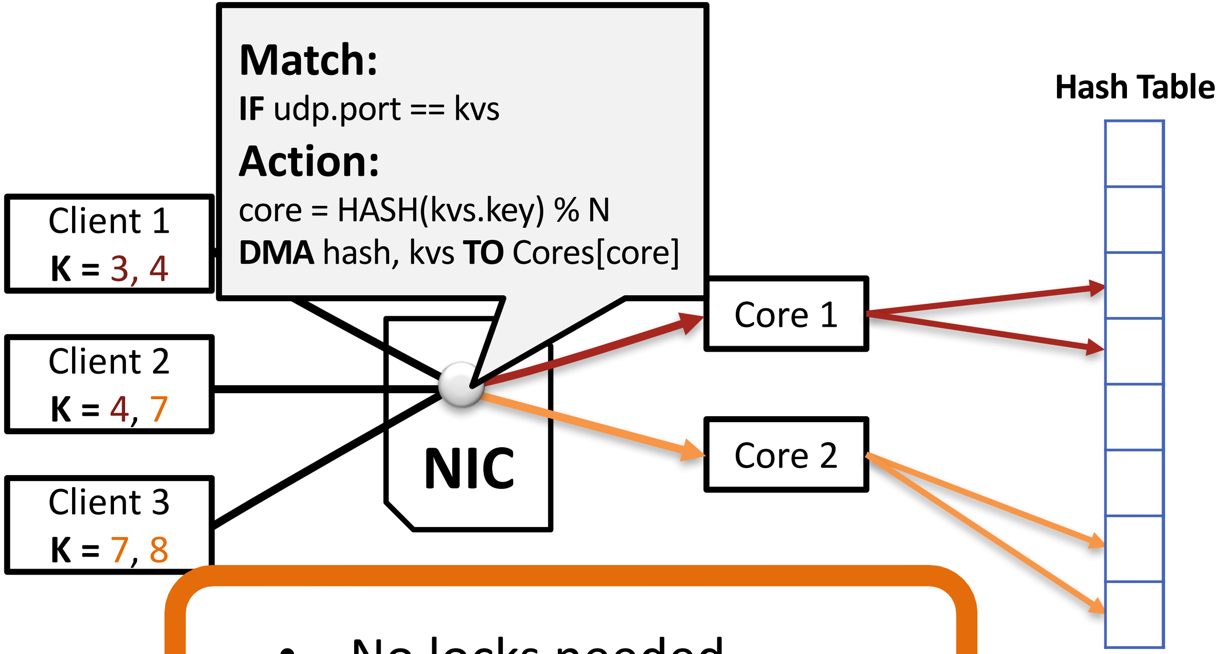
ACCELERATING PACKET-ORIENTED NETWORKING

Example: Key-Value Store



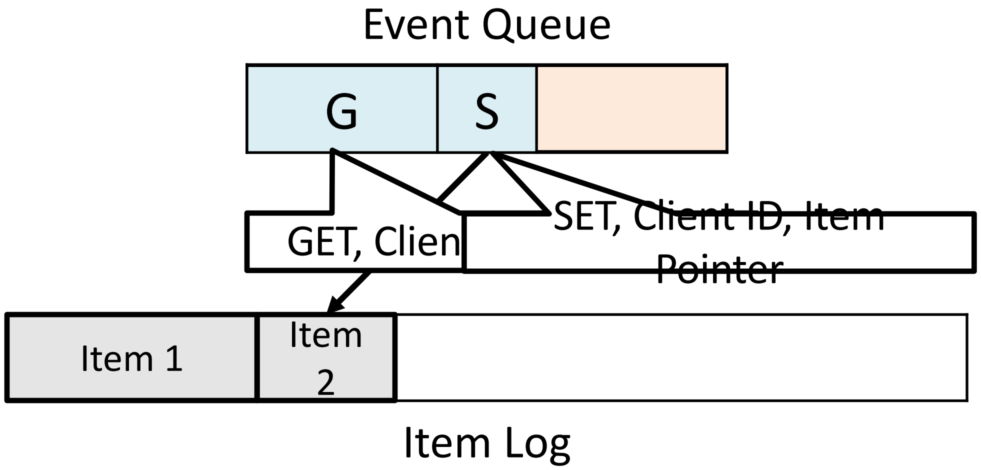
- Lock contention
- Poor cache utilization

Key-based Steering



- No locks needed
- Higher cache utilization

Custom DMA



DMA to application-level data structures

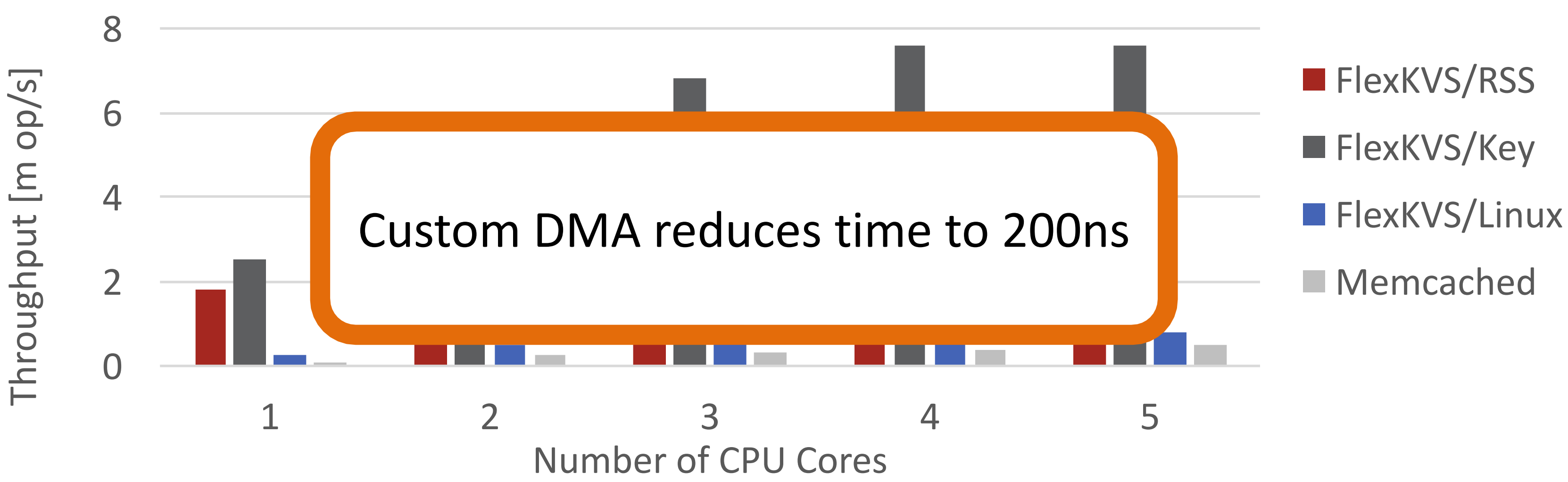
Requires packet validation and transformation

Evaluation of the Model

- Measure impact on application performance
- Key-based steering: Use NIC
- Custom DMA: Software emulation of M&A pipeline

- Workload: 100k 32B keys, 64B values, 90% GET
- 6 Core Sandy Bridge Xeon 2.2GHz, 2x10G links

Key-based steering

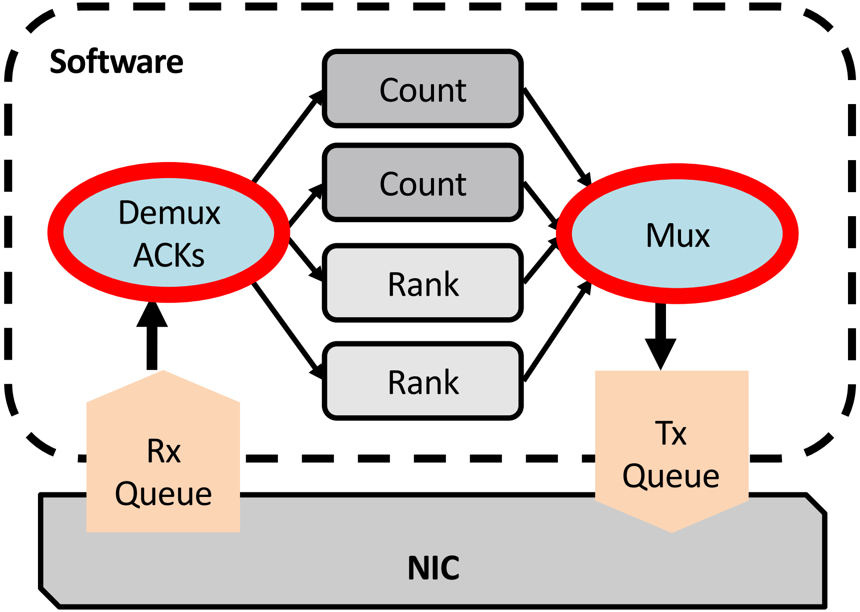


- Better scalability
  - PCIe is bottleneck for 4+ cores
- 45% higher throughput
- Processing time reduced to 310ns

Real-time Analytics System

(De-)Multiplexing threads are performance bottleneck

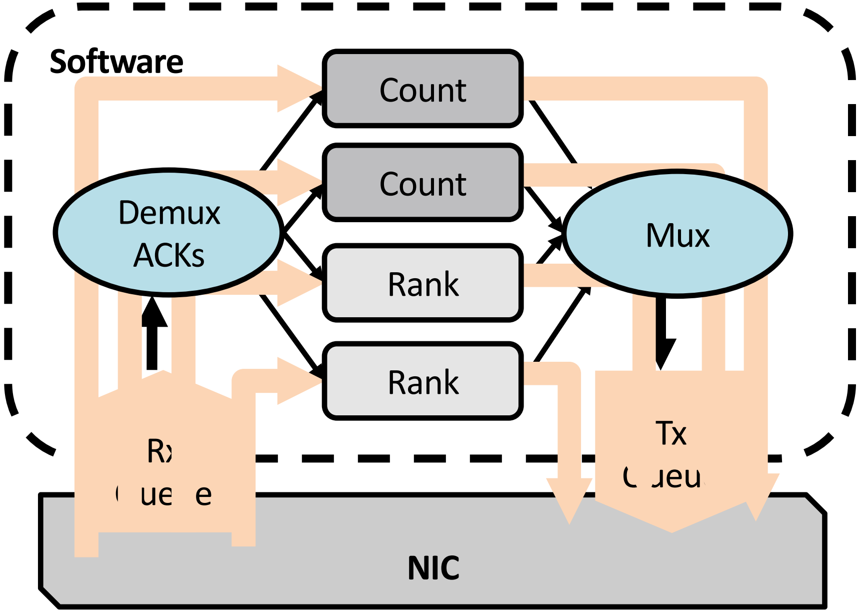
- 2 CPUs required for 10 Gb/s => 20 CPUs for 100 Gb/s



Real-time Analytics System

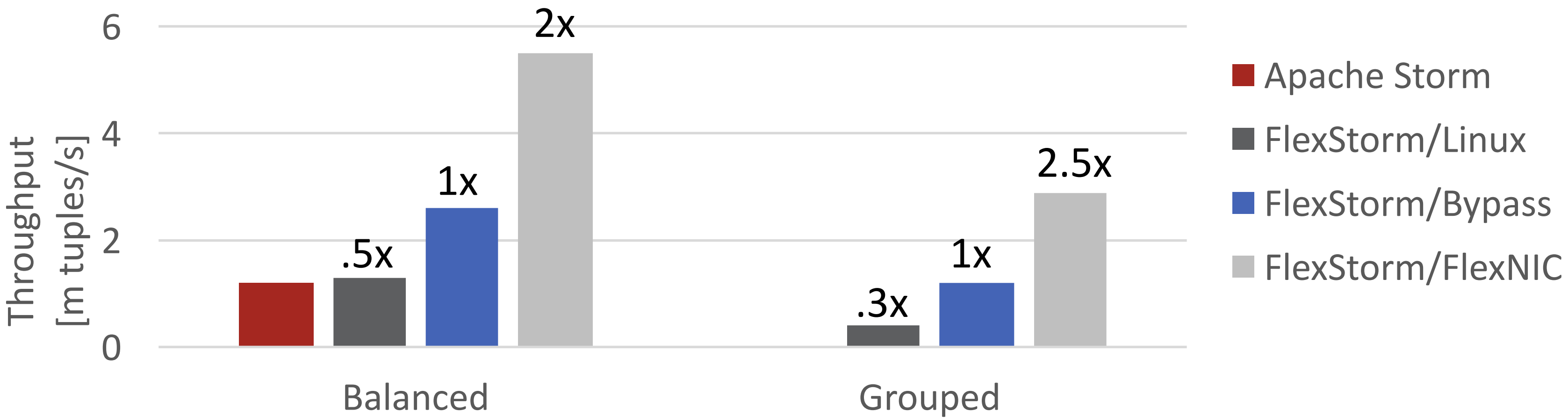
Offload (de)multiplexing and ACK generation to FlexNIC

- No CPUs needed => Energy-efficiency



Performance Evaluation

- Cluster of 3 machines
- Determine Top-n Twitter posters (real trace)
- Measure attainable throughput



Network Intrusion Detection

Snort sniffs packets and analyzes them

- Parallelized by running multiple instances
- Status quo: Receive-side scaling

FlexNIC:

- Analyze rules loaded into Snort
- Partition rules among cores to maximize caching
- Fine-grained steering to cores

Result: 1.6x higher throughput, 30% fewer cache misses

ACCELERATING STREAM-ORIENTED NETWORKING

Ongoing work: **Stream protocols**

Full TCP processing is too complex for M&A processing

- Significant connection state required
- Tricky edge cases: reordering, drops
- Complicated algorithms for congestion control

But the common case is simpler: it can be offloaded

- Reduces the critical path in software

Opportunity: Enforce correct protocol onto untrusted app

- Focus: congestion control

FlexTCP ideas

Safety critical & common processing on NIC

- Includes filtering, validating ACKs, enforcing rate limits

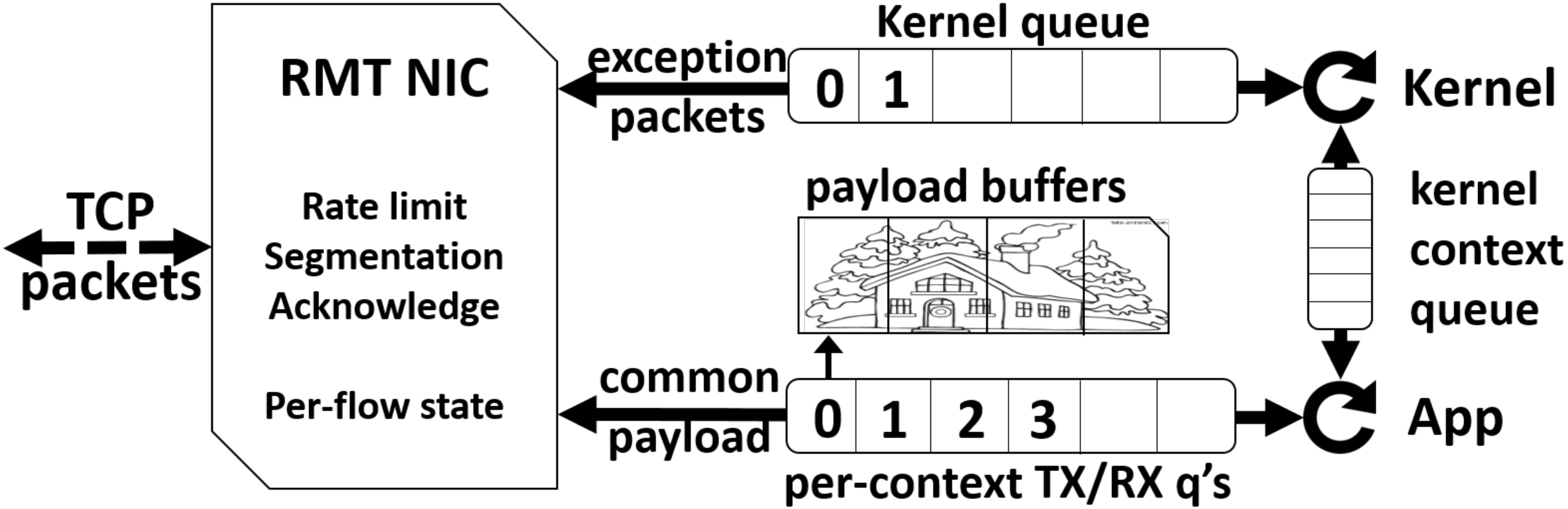
Handle all non-common cases in software

- E.g. packet drops, re-ordering, timeouts, …

Requires small per-flow state

- 64 bytes (SEQ/ACK, queues, rate-limit, …)

FlexTCP overview



Flexible congestion control offload

NIC enforces per-flow rate limits set by trusted kernel

- Flexibility to choose congestion control

Example: DCTCP

Common-case processing on NIC

- Echo ECN marks in generated ACK
- Track fraction of ECN marked packets per flow

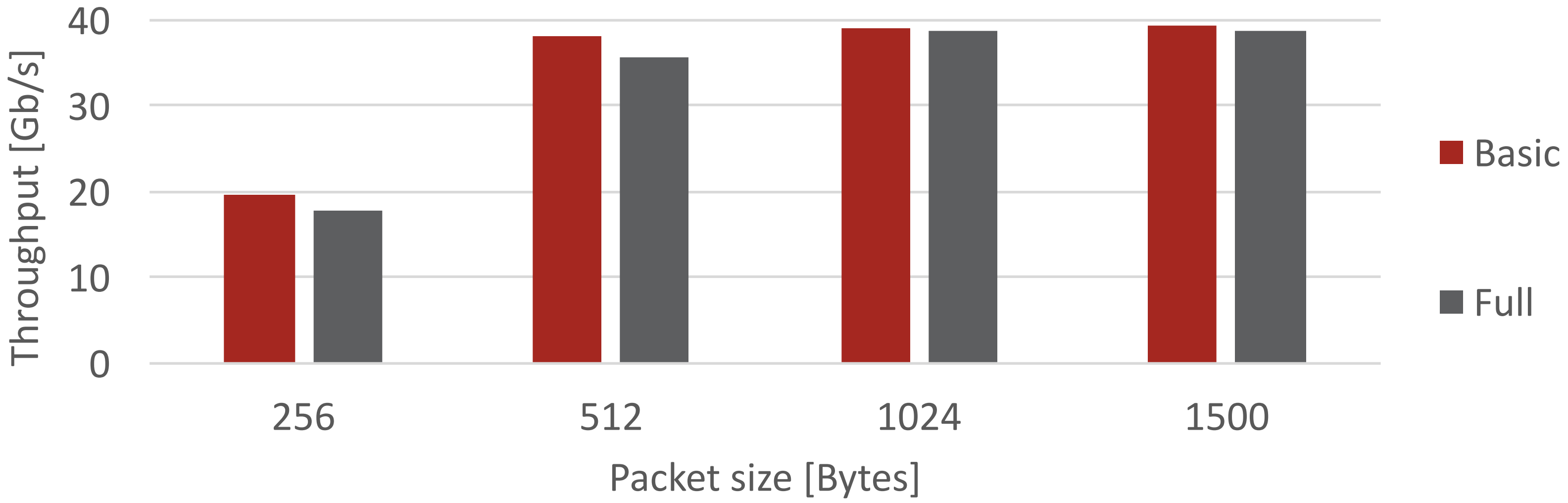
Kernel implements control policy (DCTCP)

- Use NIC-reported fraction of packets that are ECN marked
- Adapt rate limit according to DCTCP protocol

Result: Indistinguishable from pure software implementations

FlexTCP overhead evaluation

- We implemented FlexTCP in P4
- Run on Agilio-CX with null application
- Compare throughput to basic NIC (wiretest)



Summary

Networks are becoming faster, CPUs are not

- Server applications need to keep up
- Fast I/O requires efficient I/O path to application

Flexible offloads can eliminate inefficiencies

- Application control over where packets are processed
- Efficient steering, validation, transformation

Case studies: Key-value store, real-time analytics, IDS

- Up to 2.5x throughput & latency improvement vs. kernel-bypass
- Vastly more energy-efficient (no CPUs for packet processing)