Evaluating the Power of Flexible Packet Processing for Network Resource Allocation

Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson,
Changhoon Kim, Arvind Krishnamurthy, Jacob Nelson, Simon Peter

Programmable Switching Hardware

Reconfigurable network chips that process at line rate

- Not software or FPGA based routers
- Achieve high performance using a restricted computation model e.g. RMT

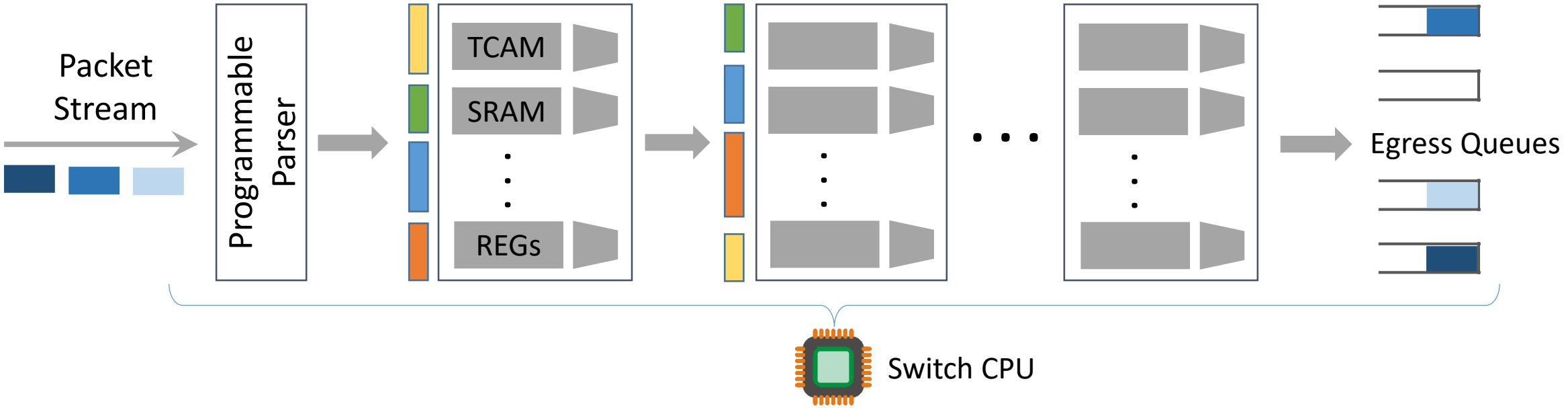
Allows operators to reconfigure switches in the field

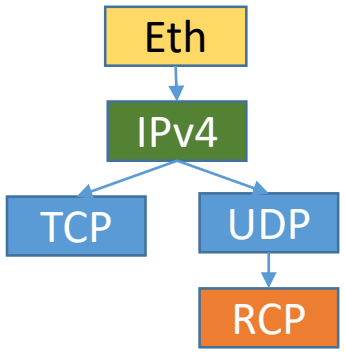
- Deploy new protocols without waiting for new hardware
- Re-allocate resources to various switch features (L2, L3 or ACLs)

Flexibility comes at a minimal hardware cost

- Commercial examples include Cavium's XPliant, Barefoot's Tofino

Features of Flexible Switches





- TCAM for arbitrary wildcard matches
- SRAM for exact/LPM lookups
- Stateful memory for counter and meters

Match

- ALUs for modifying headers and registers

Action

1. p = lookup(eth.dst_mac)
2. pkt.egress_port = p
3. counter [ipv4.src_ip] ++

Flexible Switches are not all-powerful

Processing primitives are limited

- Cannot perform arbitrary operations

Available stateful memory is constrained

- Cannot maintain significant per-flow state

Limited number of stages and limited communication across stages

- Imposes a limit on computation performed per-packet

What can we do with these switches?

- Custom routing and tunneling protocols such as VxLAN or MPLS
- Most are packet-level transformations involving static table lookups

What about network-rich protocols?

Several protocols proposed in the past require *active network elements*

- Persistent and mutable state for cross-packet transformations
- Require implementing some state machine functionality

They solve a broad class of network problems

- Congestion Control (XCP, RCP, QCN, HULL)
- Load Balancing (Hedera, CONGA, WCMP, Ananta)
- Fairness & QoS Scheduling (Seawall, FairCloud, CoDel, D^3)

Can we implement these amazing protocols on Flexible Switches?

Rest of the Talk

Our Approach

- Analyze protocols to determine their requirements
- Running example: RCP (Rate Control Protocol)

Building Blocks

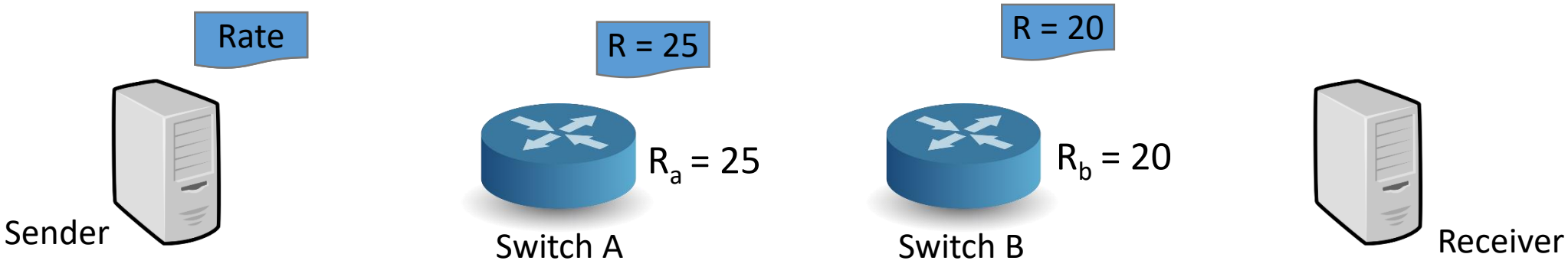
- Propose several modules that approximate some network function
- Use approximation to overcome hardware limitations

Evaluation

- Are our approximations effective and accurate enough?

Example : RCP (Rate Control Protocol)

Congestion control scheme that relies on explicit network feedback



The original algorithm computes rate periodically using,

$$R_{new} = R_{old} + \frac{\boldsymbol{\alpha} \times SpareCapacity - \boldsymbol{\beta} \times QueueSize}{Number\ of\ Flows}$$

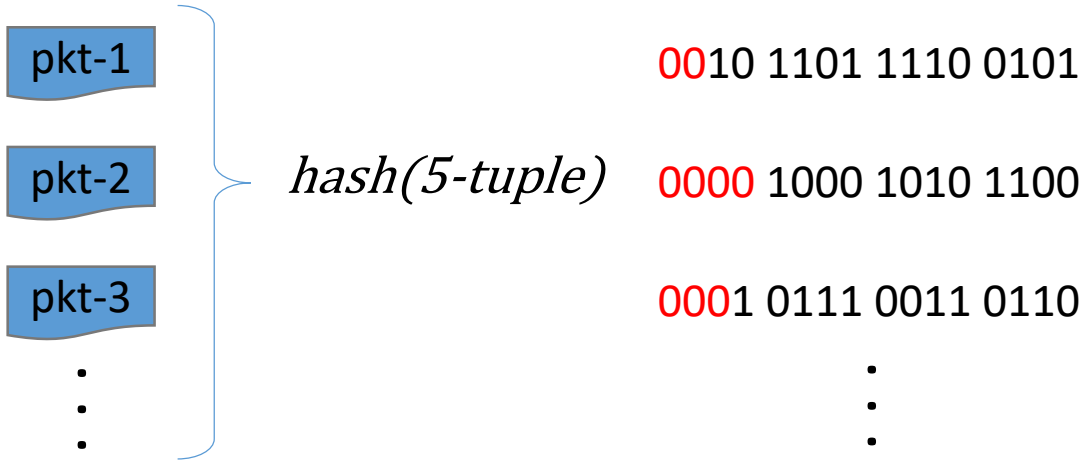
$$R_{new} = R_{old} + \frac{\boldsymbol{\alpha} \times SpareCapacity - \boldsymbol{\beta} \times QueueSize}{Number\ of\ Flows}$$

1. Measure the link utilization and queue lengths
   - Straight-forward, switches already have counters for this purpose
2. Perform multiplication and division
   - Difficult, but can be approximated using log-table lookups
3. Estimate the number of ongoing flows
   - Challenging to implement without maintaining per-flow state

Example: Cardinality Estimation

Use case: *Count the number of unique flows traversing a switch*

We extend an approach from streaming algorithms



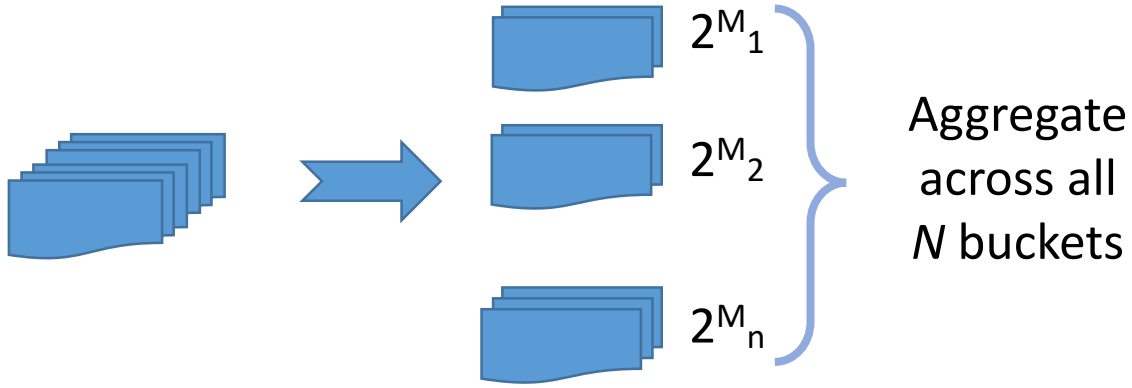
Maximum leading zeroes: 4

Estimate = $2^4 = 16$

1. For each packet, we hash the 5-tuple
2. Keep track of largest number of leading zeroes (say ***M***)
3. Estimate for number of unique elements is simply $\mathbf{2^M}$

Example: Cardinality Estimation

- The estimate is coarse-grained with high variation



- Split the input stream into multiple buckets and aggregate
- Estimate using bloom-filter counting if too few elements

Less than 5% error with 1KiB of memory for 100K flows

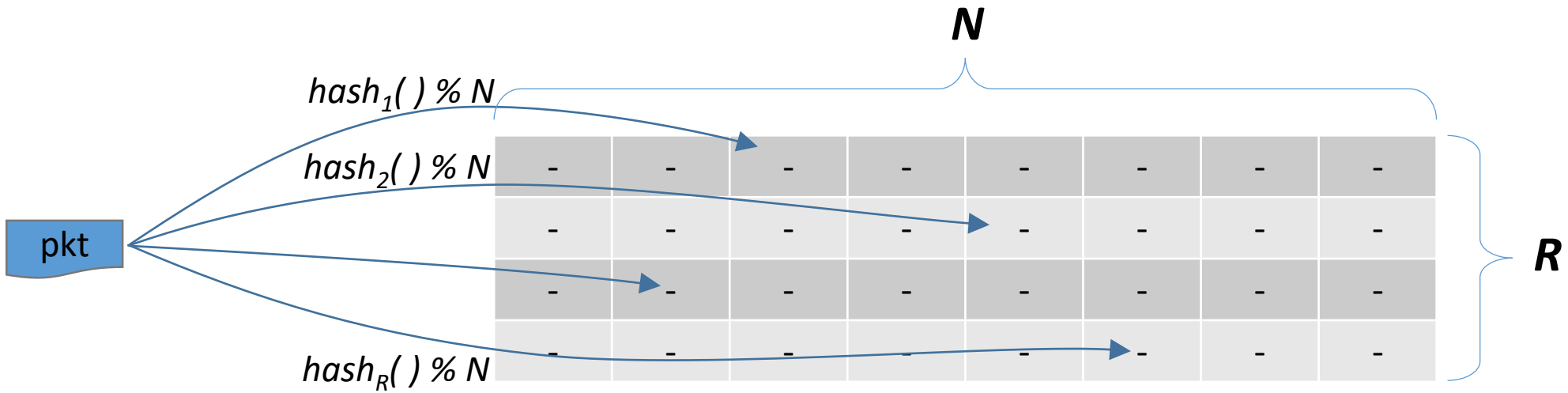
More protocols and building blocks in the paper

Functionality	Protocol	Building Blocks Required
Congestion Control	RCP	Arithmetic, Cardinality, Metering
	XCP	Arithmetic, Metering
	QCN	Arithmetic, Metering
Load Balancing	CONGA	Arithmetic, Flow Timestamps, Metering
	WCMP	Balancing
	Ananta	Flow Counters, Metering, Balancing
	Hedera	Flow Counters, Balancing
	Presto	Flow Statistics, Balancing
QoS & Fairness	Seawall	Arithmetic, Cardinality, Metering
	FairCloud	Arithmetic, Flow Counters
	CoDel	Arithmetic, Metering
	pFabric	Arithmetic, Flow Counters
Access Control	Snort IDS	Flow Counters, Cardinality
	OpenSketch	Flow Counters, Metering

Example: Approximate Flow Timestamps

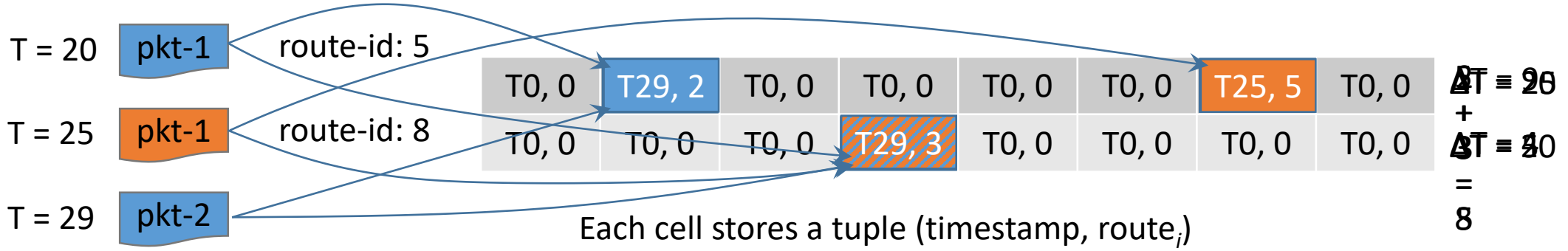
Use case: *Implement flowlet routing* (e.g., CONGA, HULA)

We use a custom sketch that stores timestamps and per-flow state



For each packet, hash the 5-tuple and update cell corresponding to $hash_i \% N$

Flowlet routing using Approximate Flow Timestamps



Flowlet Timestamps

- Flowlet is ***live***: if all cell timestamps are within timeout (say ΔT = 10)
- Update: Replace all cell timestamps with current time

Flowlet Route-id

- Sum of all cell route$_i$ is the desired route-id
- Update: Re-write all *non-live cells* such that new sum = new route-id

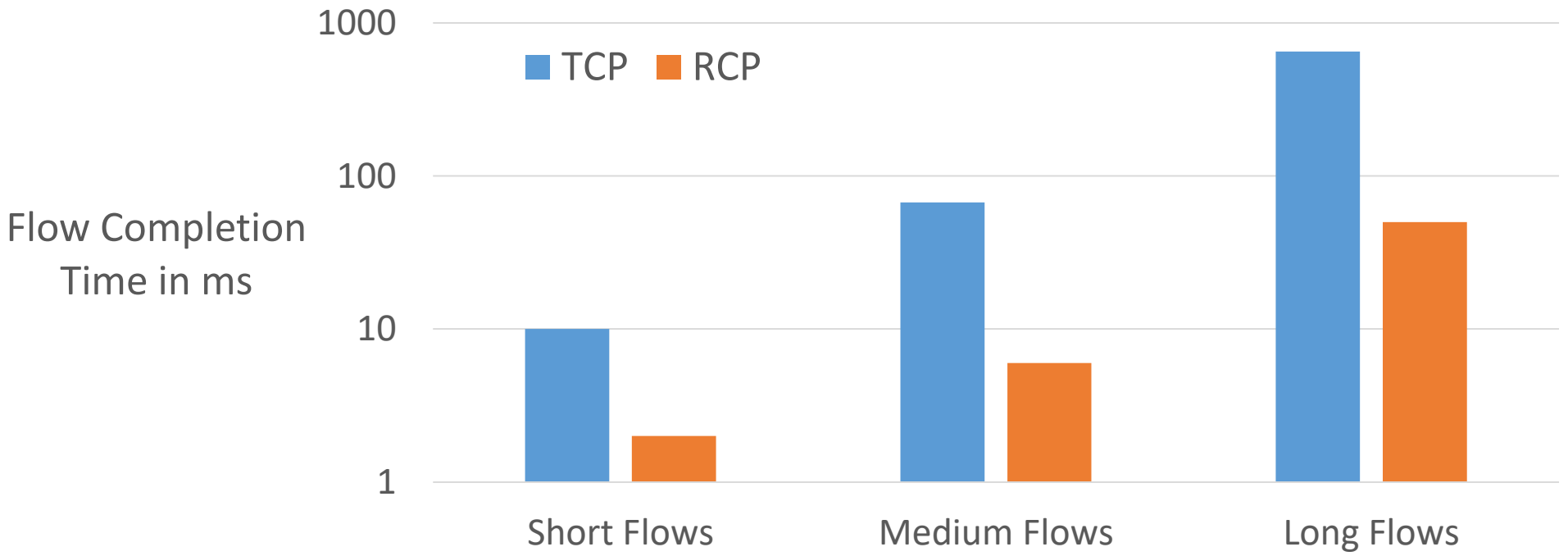
This method is always ***safe*** and maintains liveness of flowlets

Evaluation

1. Do the approximations work on real hardware ?
   - Implement RCP on Cavium XPliant switch and measure performance
2. What is the impact of approximation on protocols ?
   - Compare original and approximated versions using ns3 simulations
3. How much extra resources do approximations consume ?
   - Measure additional resource usage on top of a P4 switch

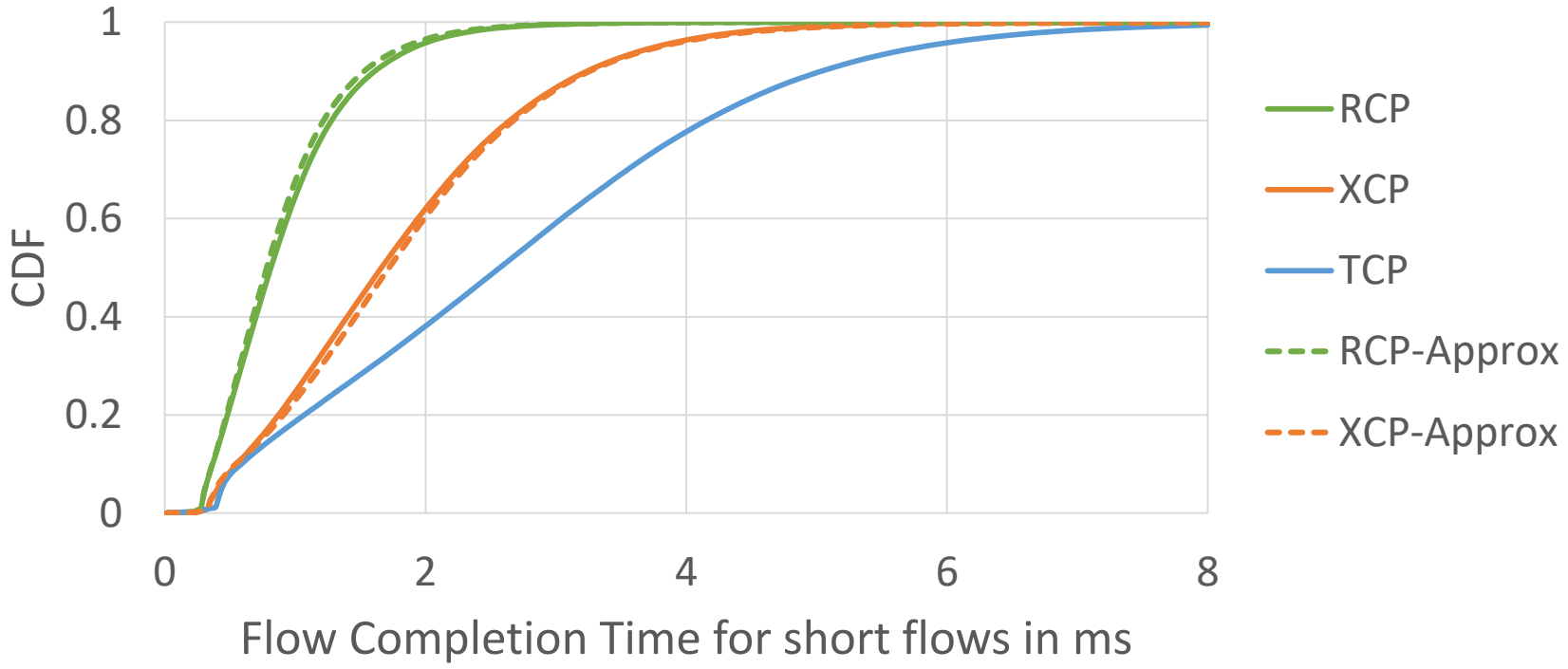
RCP implementation on a flexible switch

- Implemented on Cavium XPliant CNX880xx using our building blocks
- 2-level FatTree topology with 4 ToRs, 2 core switches using VLANs



Impact of approximation on protocols

- Implement both original and approximate protocols in ns3 simulator
- Simulate a FatTree topology with 2560 servers and 112 switches



Resource usage of approximate protocols

- Implement protocols in P4 on open-source switch.p4
- Compile to a baseline switch based on Tofino hardware model
  - L2 switching, L3 routing, LAGs, ECMP, VxLAN, GRE, Geneve, etc

Resource	Baseline	RCP		XCP		CONGA	
Packet Header Vector	187	191	+ 2%	195	+ 4%	199	+ 6%
Pipeline Stages	9	10	+ 11%	9	+ 0%	11	+ 22%
Match Crossbar	462	473	+ 2%	471	+ 2%	478	+ 3%
Hash Bits	1050	1115	+ 6%	1058	+ 1%	1137	+ 8%
SRAM	165	175	+ 6%	172	+ 4%	213	+ 29%
TCAM	43	44	+ 2%	45	+ 5%	44	+ 2%
ALU Instructions	83	88	+ 6%	92	+ 11%	98	+ 18%

Summary

- Programmable switches can implement a broad class of protocols
- Overcome hardware constraints using approximation techniques
- Approximations are effective and accurate enough
  - Demonstrate benefits on real hardware
  - Simulations show protocol performance largely unaffected by approximations
- On-going work: design new protocols tailored to flexible switches