Advanced Computer Networks

In-Network Computing

Lin Wang
Fall 2021, Period 2

VU VRIJE UNIVERSITEIT AMSTERDAM

Course outline

Warm-up

- Introduction (history, principles)
- Networking basics
- Networking data structures and algorithms
- Network transport

Data centers

- Data center networking
- Data center transport

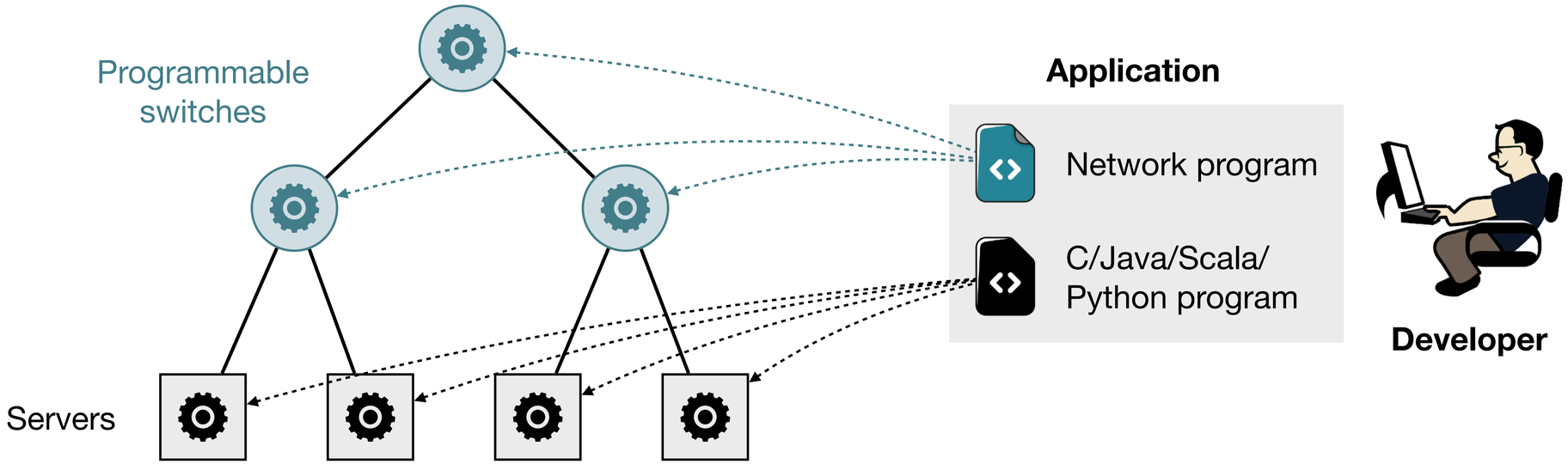
Programmability

- Software defined networking
- Network automation
- Network function virtualization
- Programmable data plane

Application

- Network monitoring
- **In-network computing**
- Machine learning for networking

In-network computing



In-network computing: performing application-specific computations “in the network” on the path between data sources and sinks, leveraging modern programmable switches

Learning objectives

How to implement an in-network caching service?

How to implement an in-network coordination service?

How to implement an in-network caching service?

Key-value storage

Store, retrieve, manage key-value objects

- Critical building block for large-scale cloud services
- Need to meet aggressive latency and throughput objectives efficiently

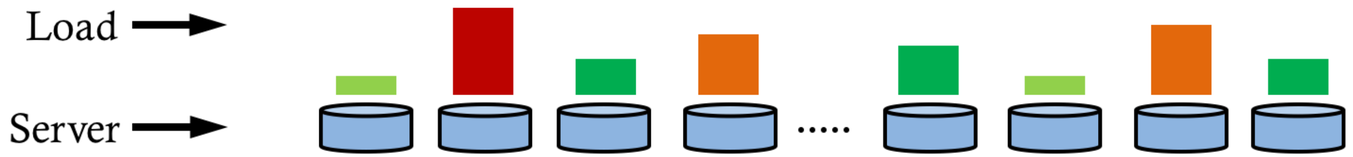
Target workloads

- Small objects
- Read intensive
- **Highly skewed and dynamic key popularity**

Challenge

Highly skewed and rapidly changing workloads

Requirement: high throughput, low (tail) latency



How to provide effective dynamic load balancing?

Opportunity

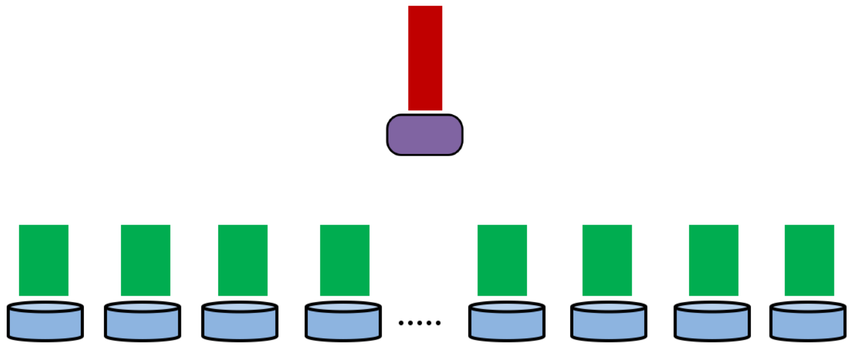
Fast, small cache can ensure load balancing

Cache O(NlogN) hottest items

- E.g., 10,000 hot objects

N: number of servers

- E.g., 100 backend servers with 100 billion items

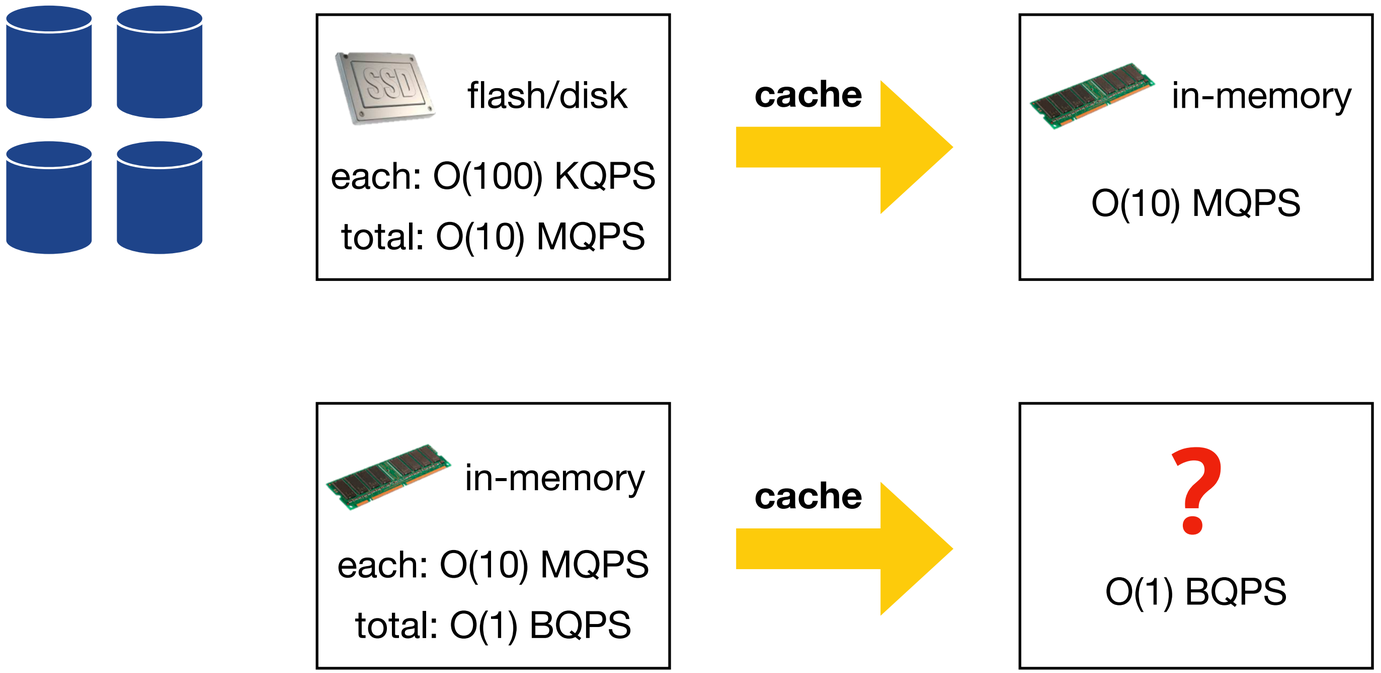


Requirement: cache throughput >= backend aggregate throughput

Bin Fan, Hyeontaek Lim, David G. Andersen, Michael Kaminsky. Small cache, big effect: provable load balancing for randomly partitioned cluster services. ACM SoCC, 2011.

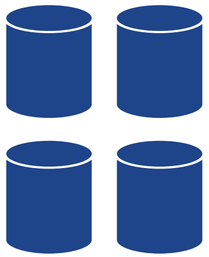
How to build the cache?

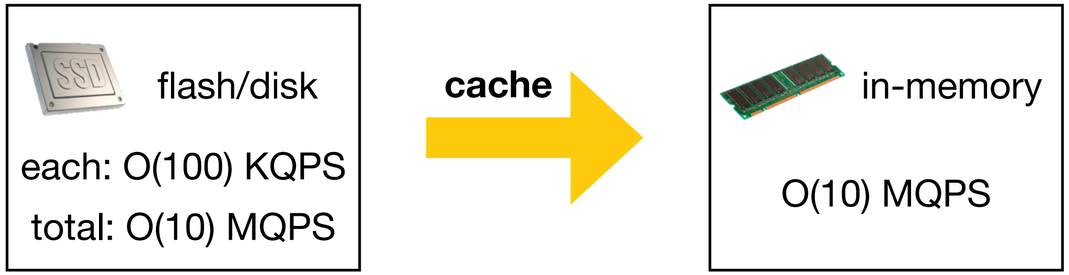
Cache needs to provide the aggregate throughput of the storage layer

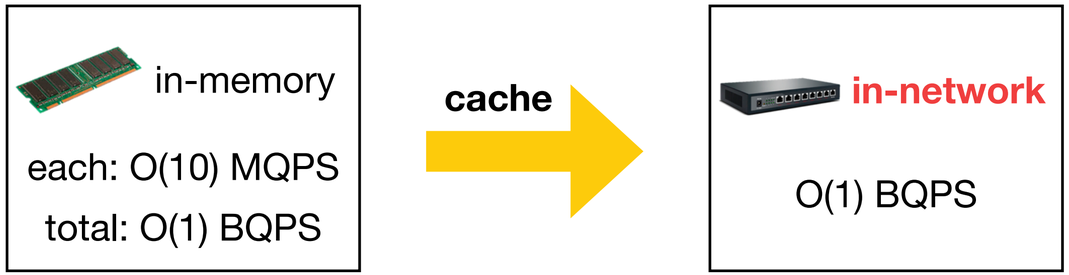


In-Network Caching

Cache needs to provide the aggregate throughput of the storage layer



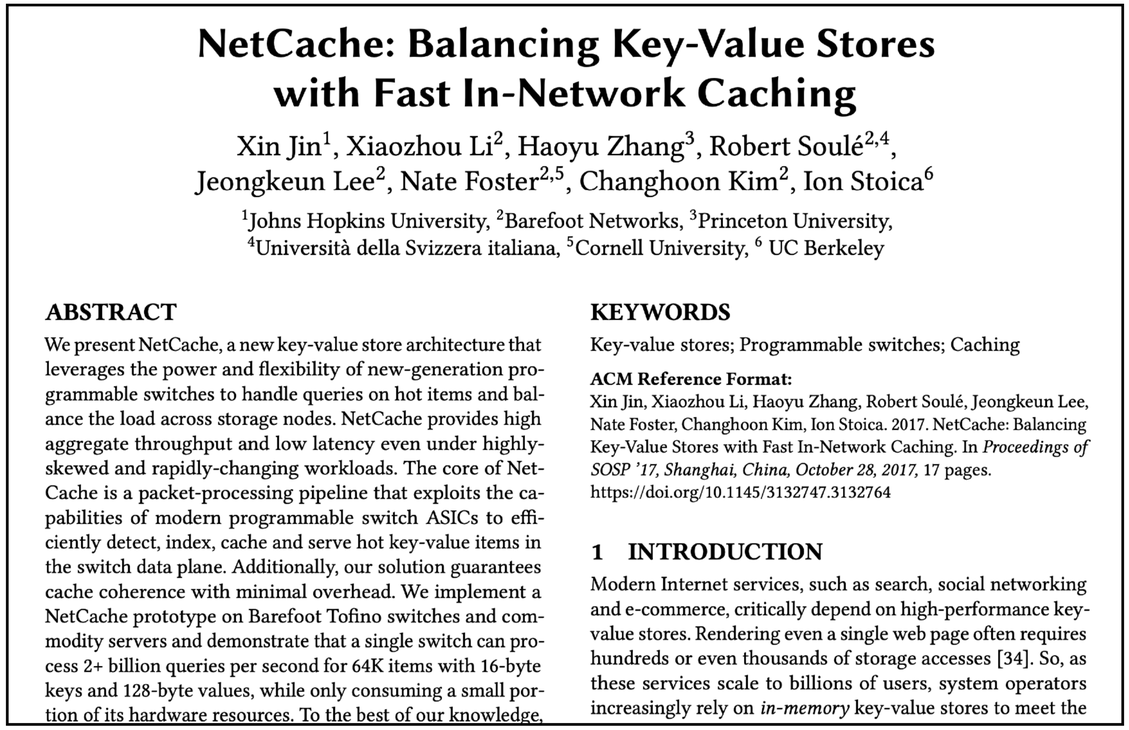




Limited on-chip memory?
But we only need to cache O(NlogN) small items!

In-Network Caching

Key-value caching in network ASIC at line rate



NetCache: Balancing Key-Value Stores with Fast In-Network Caching

Xin Jin[1], Xiaozhou Li[2], Haoyu Zhang[3], Robert Soulé[2,4], Jeongkeun Lee[2], Nate Foster[2,5], Changhoon Kim[2], Ion Stoica[6]

[1]Johns Hopkins University, [2]Barefoot Networks, [3]Princeton University, [4]Università della Svizzera italiana, [5]Cornell University, [6] UC Berkeley

ABSTRACT

We present NetCache, a new key-value store architecture that leverages the power and flexibility of new-generation programmable switches to handle queries on hot items and balance the load across storage nodes. NetCache provides high aggregate throughput and low latency even under highly-skewed and rapidly-changing workloads. The core of NetCache is a packet-processing pipeline that exploits the capabilities of modern programmable switch ASICs to efficiently detect, index, cache and serve hot key-value items in the switch data plane. Additionally, our solution guarantees cache coherence with minimal overhead. We implement a NetCache prototype on Barefoot Tofino switches and commodity servers and demonstrate that a single switch can process 2+ billion queries per second for 64K items with 16-byte keys and 128-byte values, while only consuming a small portion of its hardware resources. To the best of our knowledge,

KEYWORDS

Key-value stores; Programmable switches; Caching

ACM Reference Format:

Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of SOSP '17, Shanghai, China, October 28, 2017,* 17 pages.
https://doi.org/10.1145/3132747.3132764

1 INTRODUCTION

Modern Internet services, such as search, social networking and e-commerce, critically depend on high-performance key-value stores. Rendering even a single web page often requires hundreds or even thousands of storage accesses [34]. So, as these services scale to billions of users, system operators increasingly rely on *in-memory* key-value stores to meet the

ACM SOSP 2017

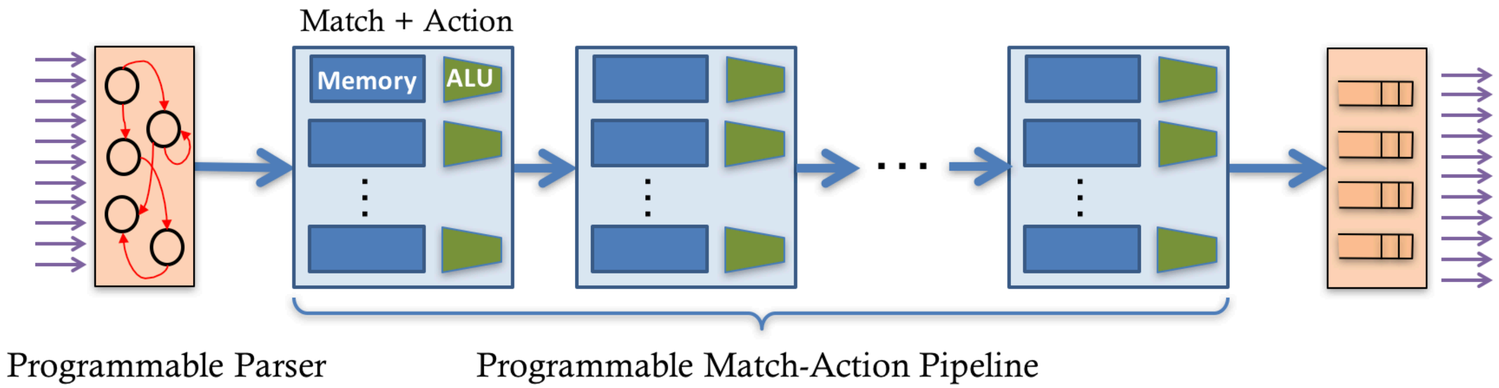
Recall PISA

Programmable parser

- Extra packet header and converts packet data into metadata

Programmable match-action pipeline

- Operate on metadata and update memory states



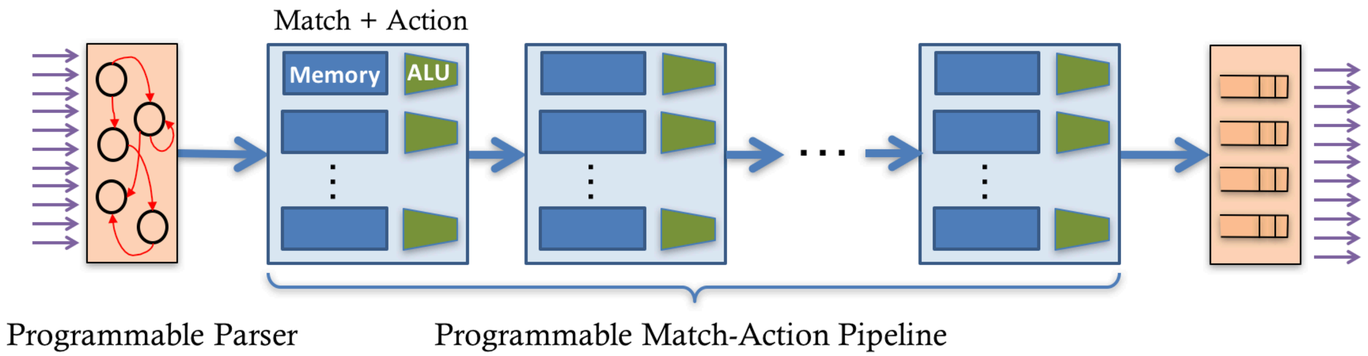
Use PISA for key-value store

Programmable parser

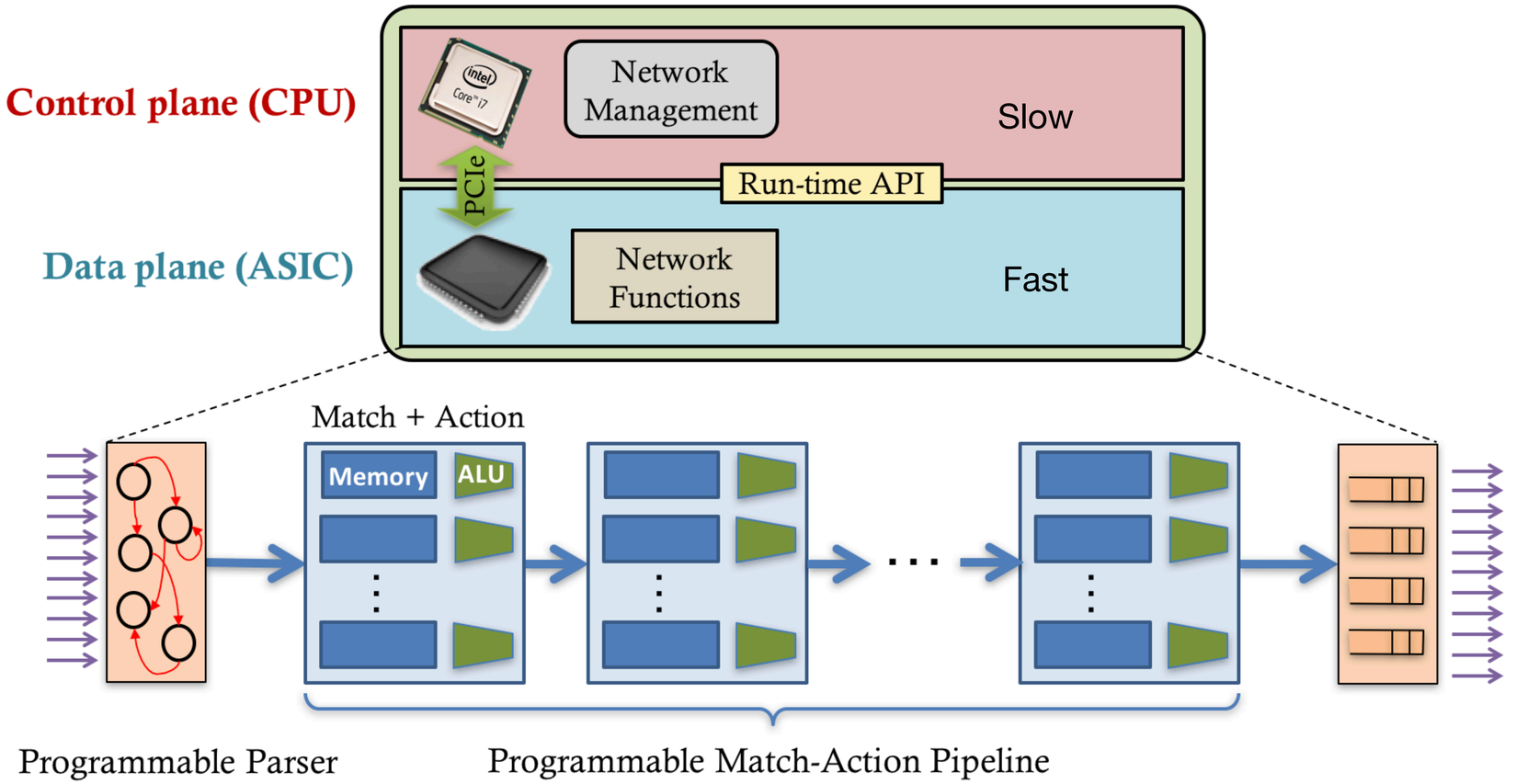
- Parse custom key-value fields in the packet header

Programmable match-action pipeline

- Read and update key-value data
- Provide query statistics for cache updates



PISA switch architecture



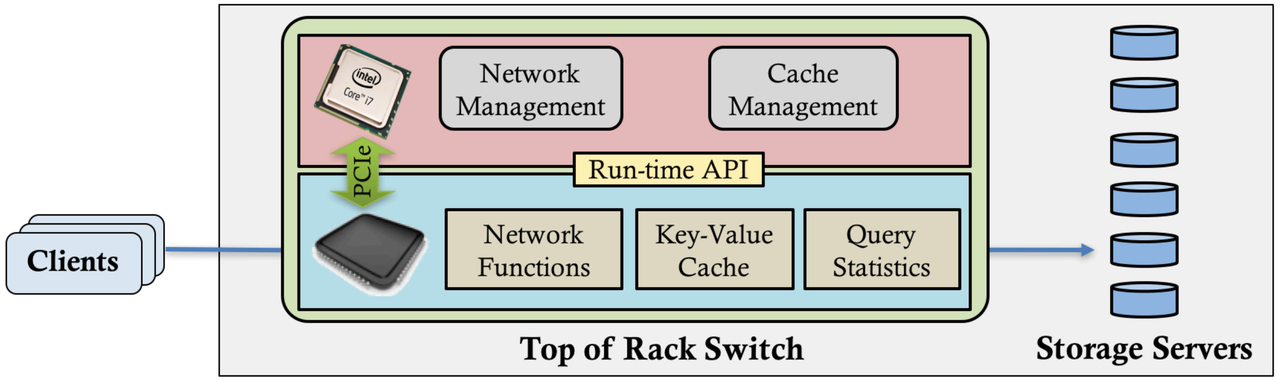
NetCache rack-scale architecture

Switch data plane

- **Key-value** store to serve queries for cached keys
- **Query statistics** to enable efficient cache updates

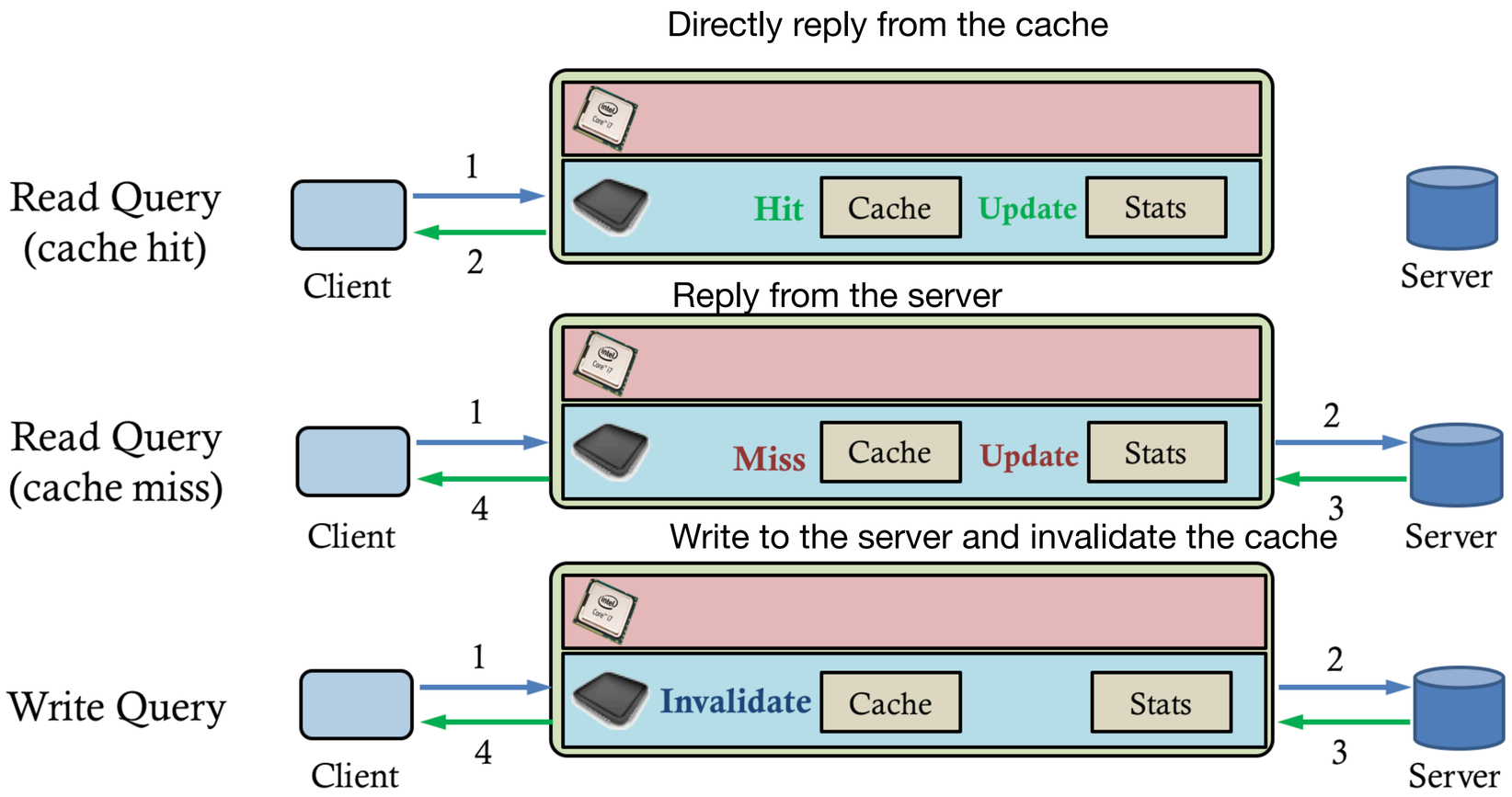
Switch control plane

- Insert hot items into the cache and evict less popular items
- Manage memory allocation for on-chip key-value store



Data plane query handling

Three cases: read (hit), read (miss), write



In-Network Caching

Key-value caching in network ASIC at line rate?!

Research questions to answer

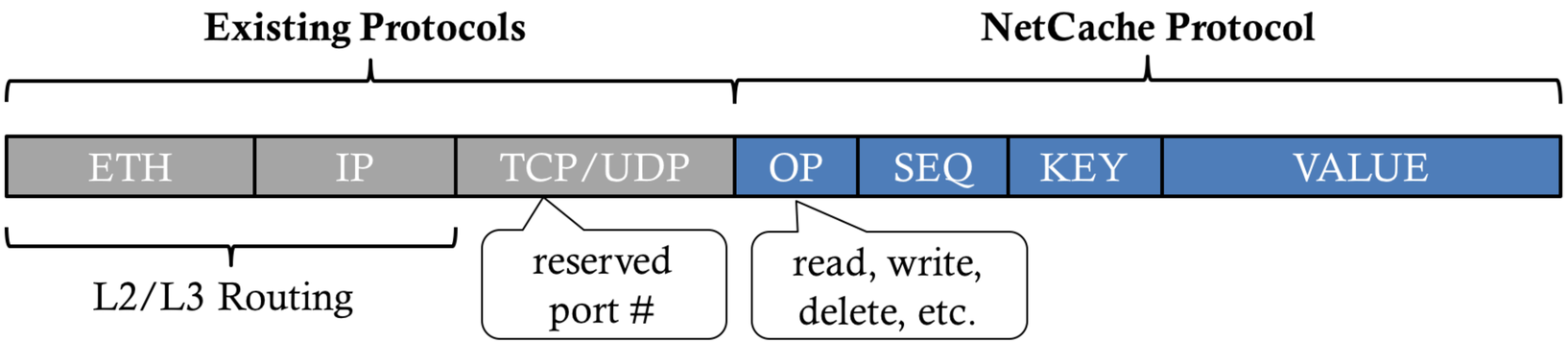
How to identify application-level packet fields?

How to store and serve variable-length data on switches?

How to efficiently keep the cache up-to-date?

NetCache packet format

Application-layer protocol: compatible with existing L2-L4 layers



Only the top-of-rack switch needs to parse NetCache fields

In-Network Caching

Key-value caching in network ASIC at line rate?!

Research questions to answer

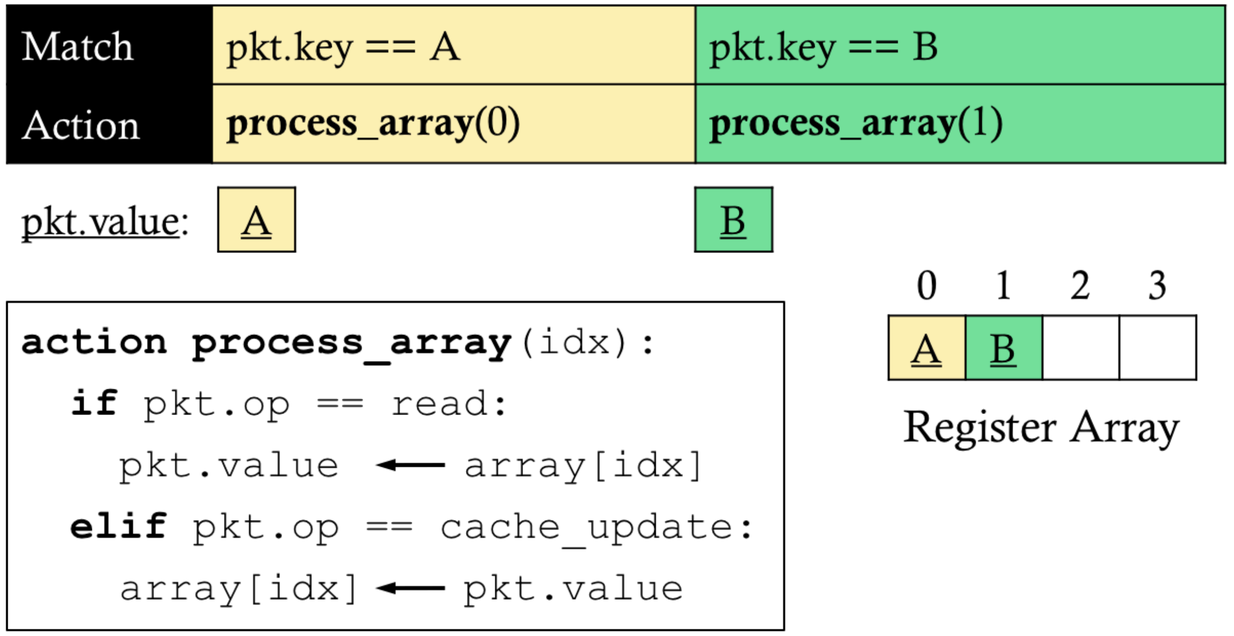
How to identify application-level packet fields?

How to store and serve variable-length data on switches?

How to efficiently keep the cache up-to-date?

Use register array

For fixed-length items, use the index to fetch items from the register array

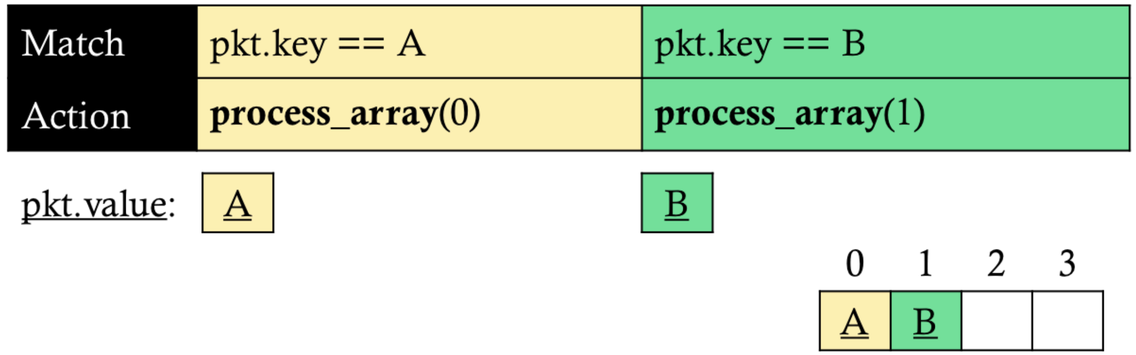


Challenges with variable length

No loop or string due to strict timing requirements

Need to optimize hardware resources consumption

- Number of table entries
- Size of action data from each entry
- Size of intermediate metadata across tables



Potential solutions

Solution 1: use action data to hold the indices

RA

Match	pkt.key == A
Action	process_array([2,3,...])

Problem: number of lookups in one register array (RA) is limited

Solution 2: use multiple register arrays (RAs) with the same index

RA1

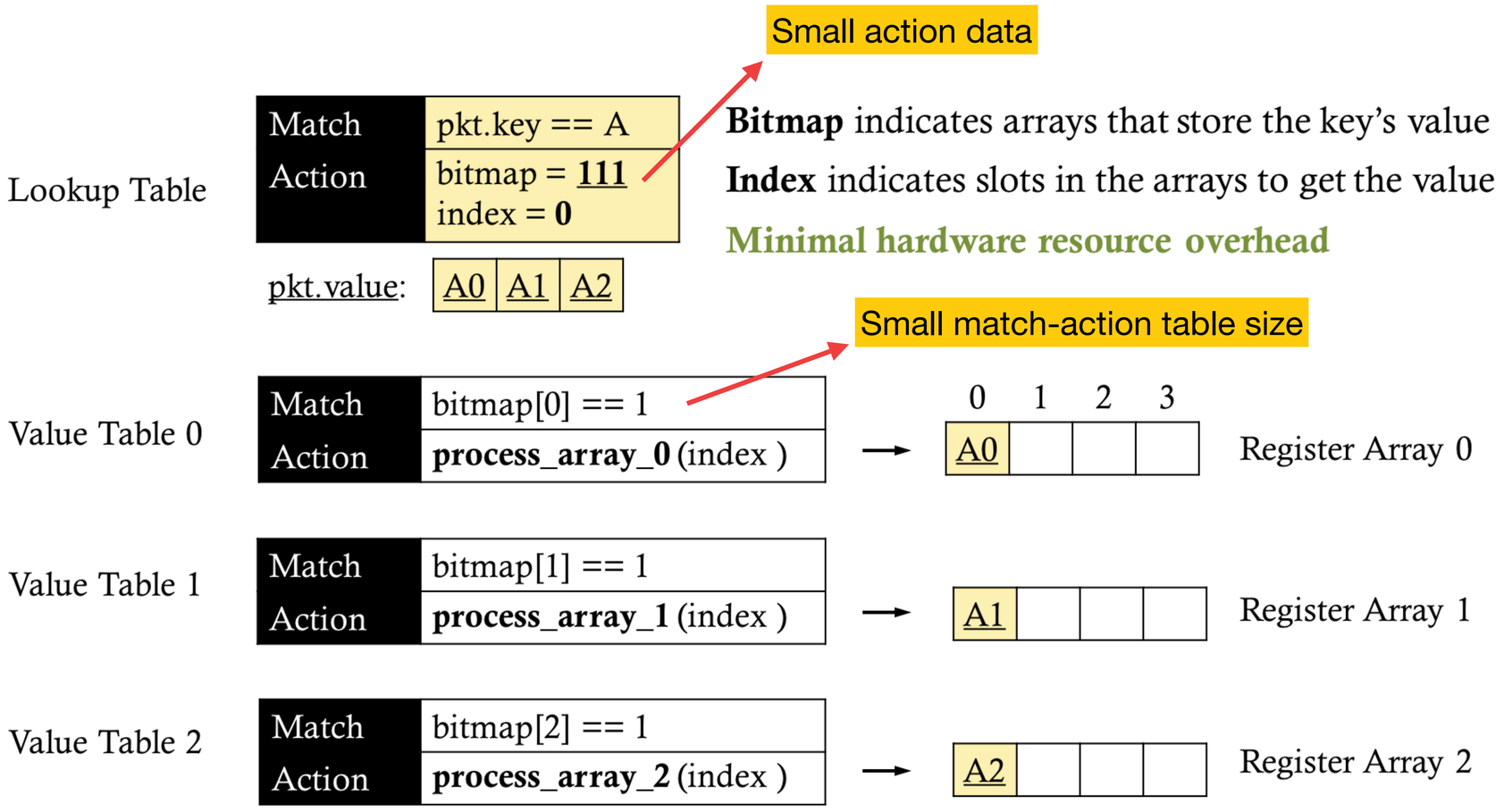
Match	pkt.key == A
Action	process_array(2)

RA2

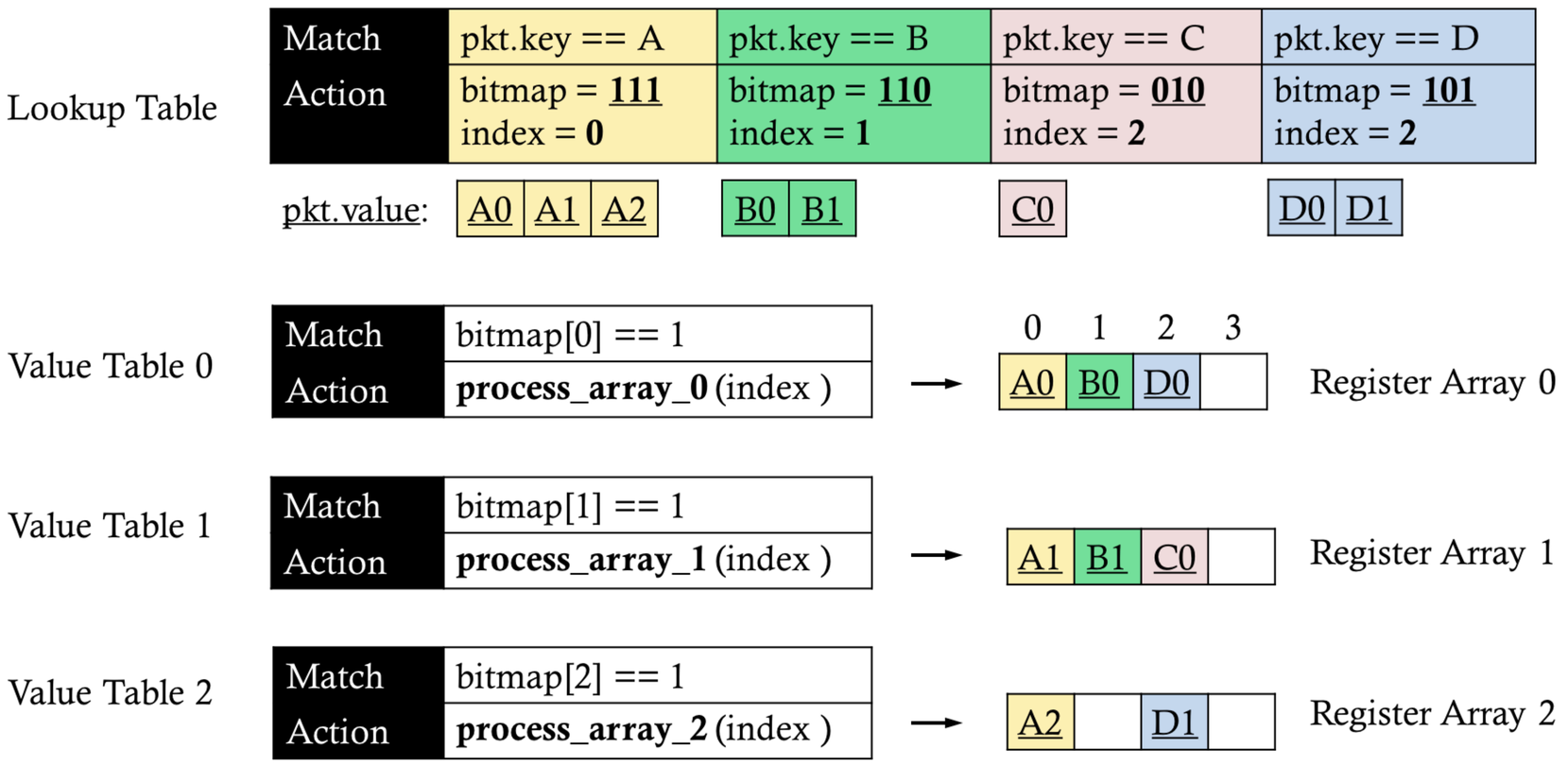
Match	pkt.key == A
Action	process_array(2)

Problem: too many match action table entries

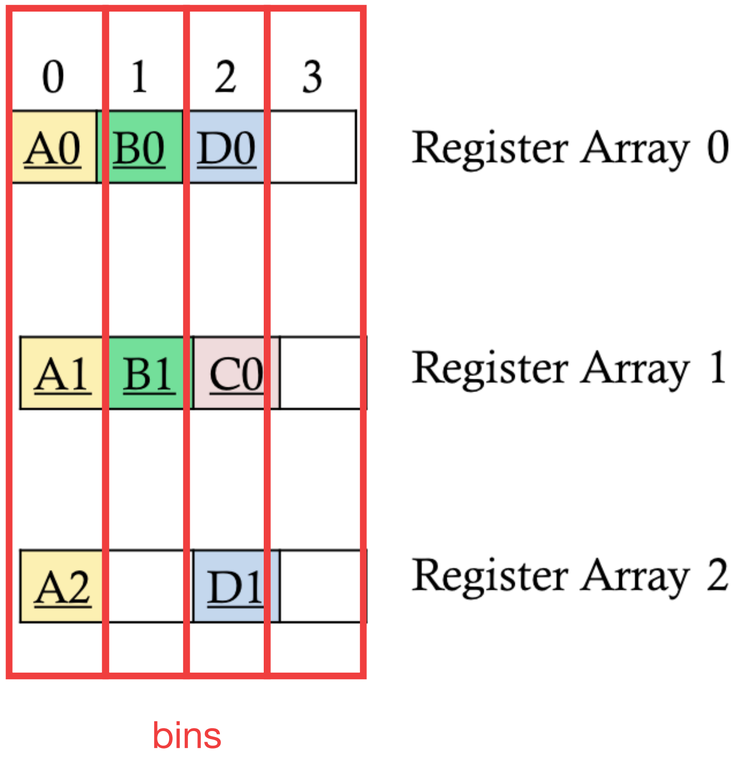
NetCache: two-level lookup



Combine outputs from multiple arrays



Memory management



Solving a bin-packing problem: use First-Fit heuristics

In-Network Caching

Key-value caching in network ASIC at line rate?!

Research questions to answer

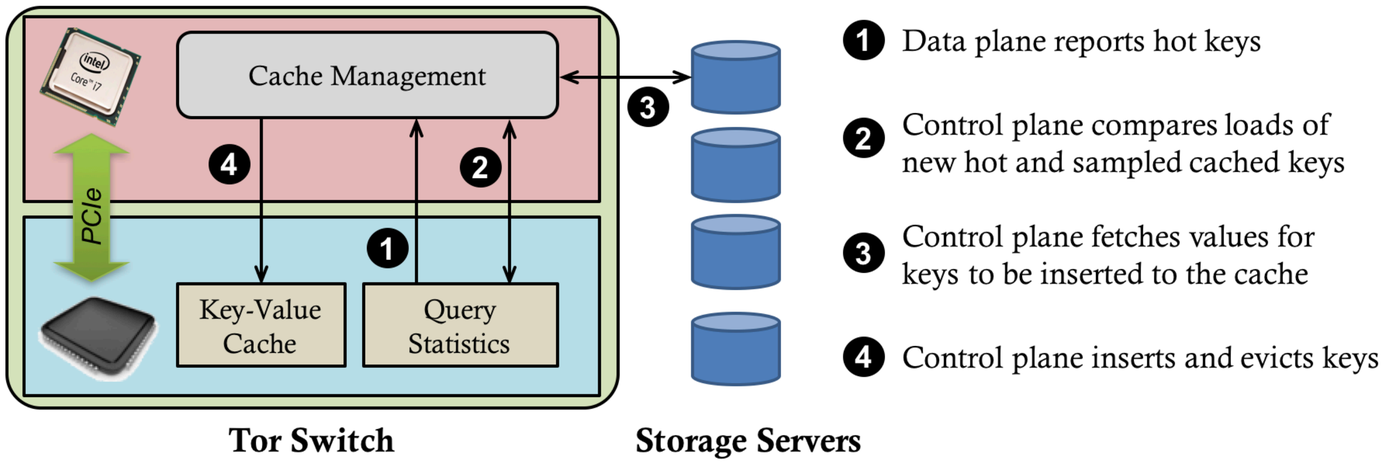
How to identify application-level packet fields?

How to store and serve variable-length data on switches?

How to efficiently keep the cache up-to-date?

Cache insertion and eviction

Goal: react quickly and effectively to workload changes with minimal updates



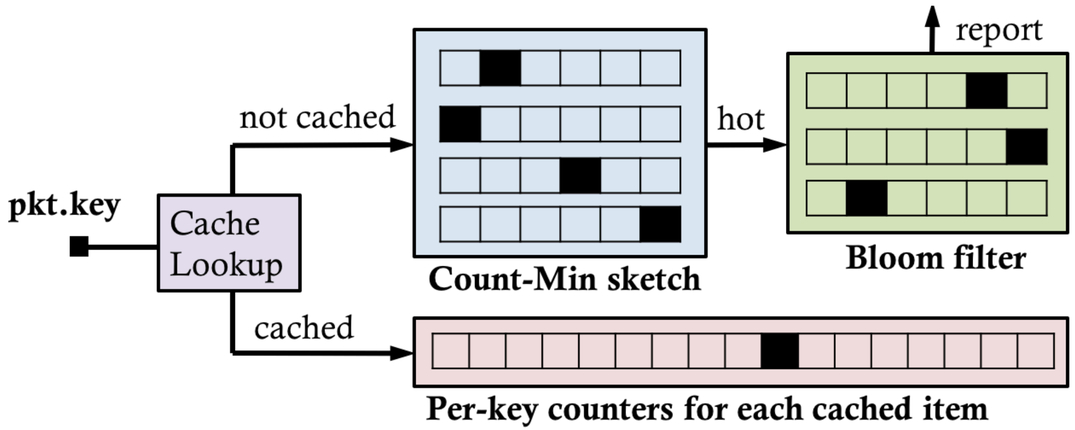
Challenge: cache the hottest O(NlogN) items with limited insertion rate

Query statistics

Cached key: per-key counter array

Uncached key:

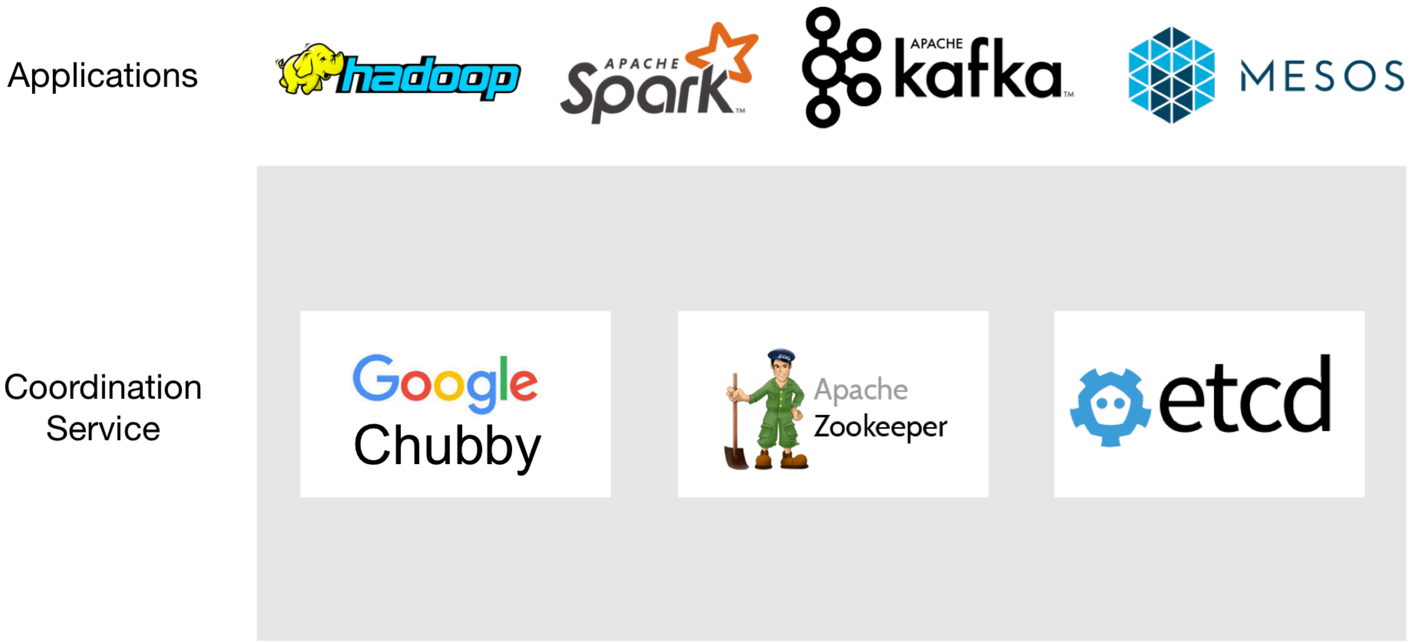
- Count-min sketch: report new hot keys
- Bloom filter: remove duplicated hot key reports



How to implement an in-network coordination service?

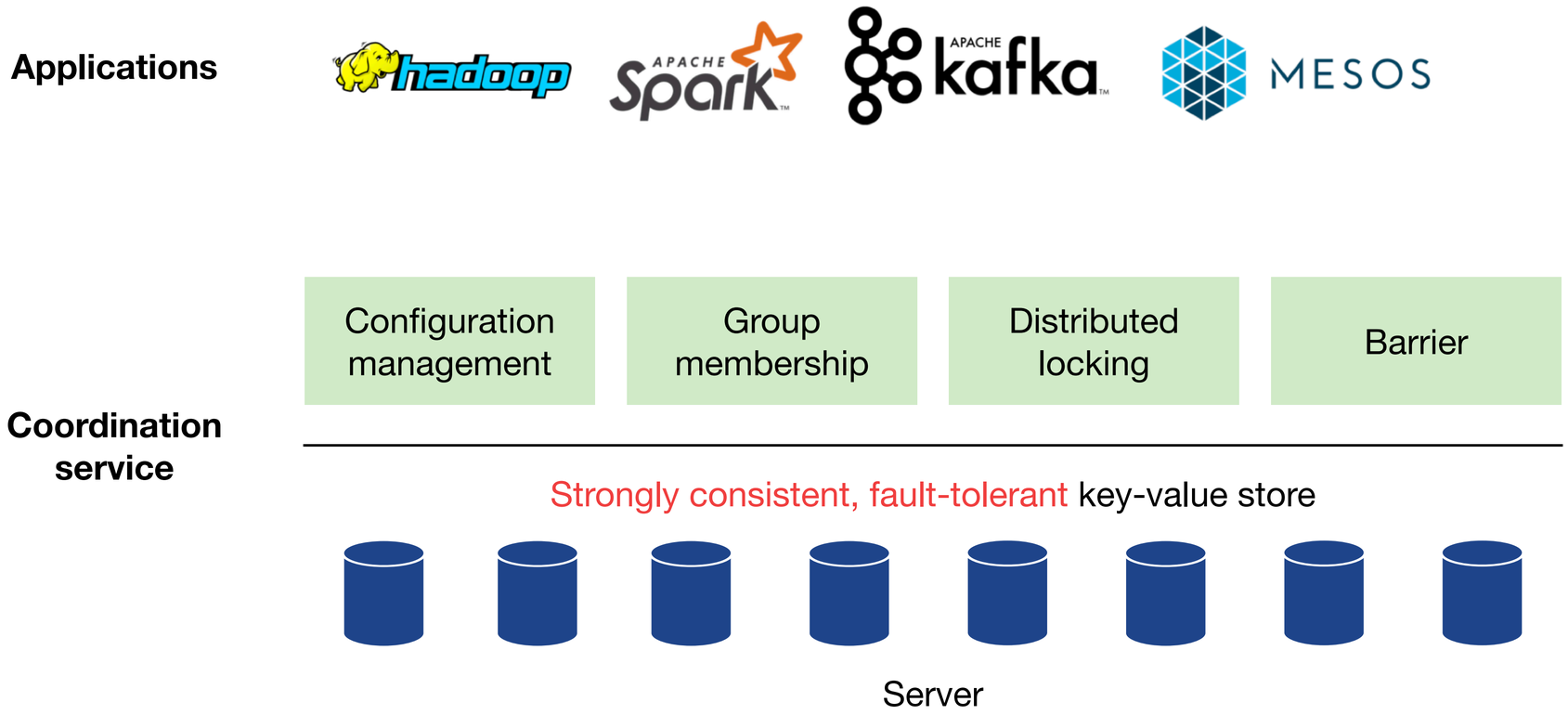
Coordination service

Fundamental building block of distributed systems, e.g., the cloud



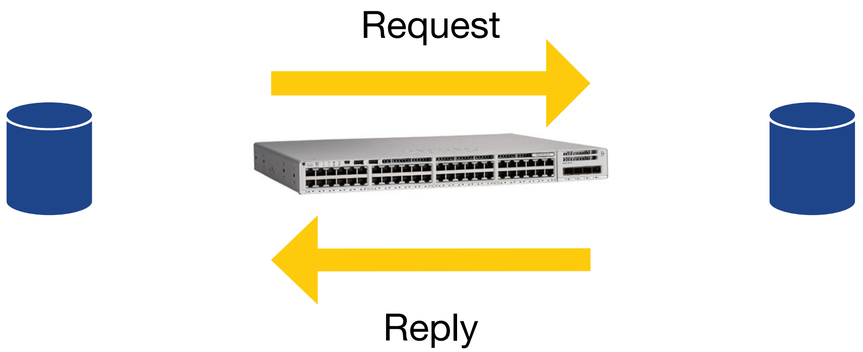
Coordination service

Can be built on strongly consistent, fault-tolerant storage services



Workflow of coordination service

Multiple message exchanges between servers on the network

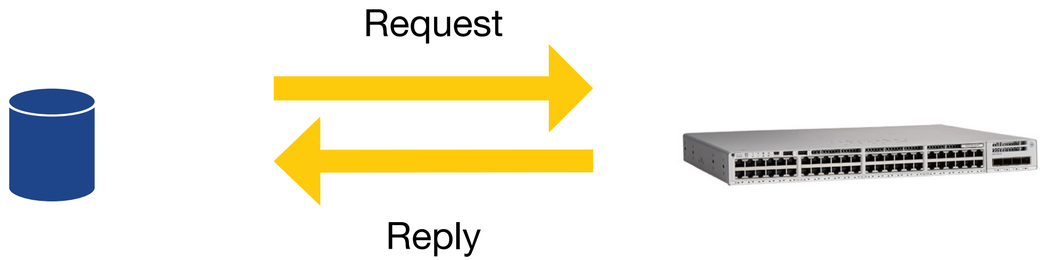


Throughput: limited by the server NIC throughput

Latency: at least one RTT, typically a few RTTs

In-network coordination

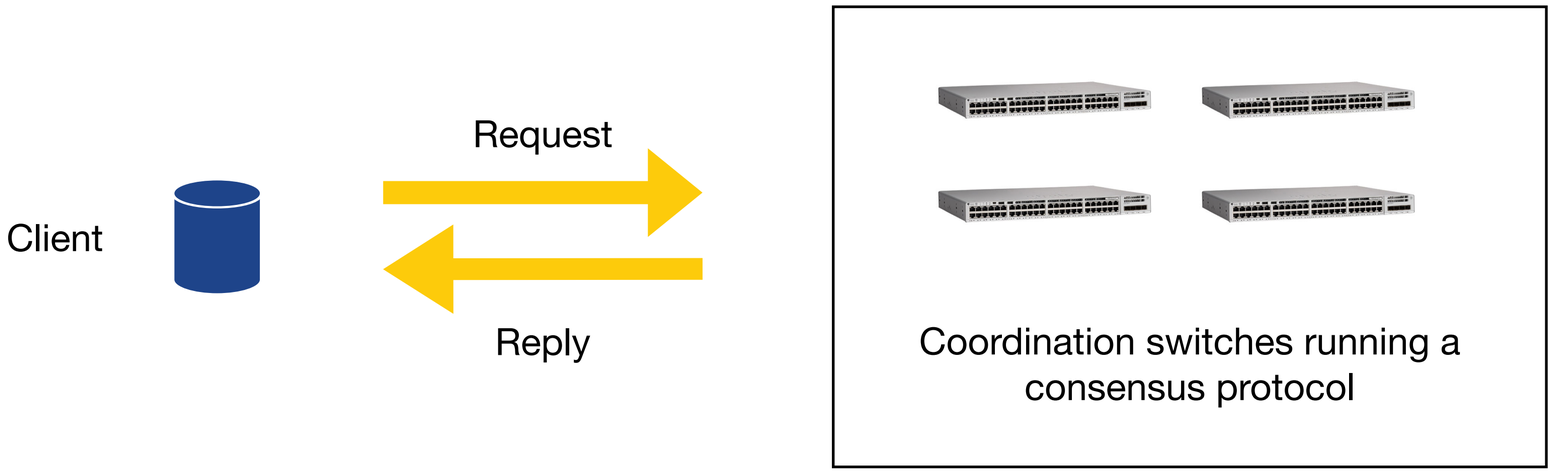
Motivation: in-network coordination is communication-heavy, not computation-heavy



	Server	Switch
Example	[NetBricks, OSDI'16]	**Barefoot Tofino**
Packets per second	30 million	**A few billion**
Bandwidth	10-100 Gbps	**6.5 Tbps**
Processing delay	10-100 us	**< 1 us**

In-network coordination

Use a set of coordination switches to run a consensus protocol



Throughout: switch throughput

Latency: sub-RTT

NetChain: Scale-Free Sub-RTT Coordination

Xin Jin[1], Xiaozhou Li[2], Haoyu Zhang[3], Nate Foster[2,4],
Jeongkeun Lee[2], Robert Soulé[2,5], Changhoon Kim[2], Ion Stoica[6]

[1]*Johns Hopkins University,* [2]*Barefoot Networks,* [3]*Princeton University,*
[4]*Cornell University,* [5]*Università della Svizzera italiana,* [6] *UC Berkeley*

Abstract

Coordination services are a fundamental building block of modern cloud systems, providing critical functionalities like configuration management and distributed locking. The major challenge is to achieve low latency and high throughput while providing strong consistency and fault-tolerance. Traditional server-based solutions require multiple round-trip times (RTTs) to process a query. This paper presents NetChain, a new approach that provides scale-free sub-RTT coordination in datacenters. NetChain exploits recent advances in programmable switches to store data and process queries entirely in the network data plane. This eliminates the query processing at coordination servers and cuts the end-to-end latency to as little as half of an RTT—clients only experience processing delay from their own software stack plus network delay, which in a datacenter set-

DrTM [6], which can process hundreds of millions of transactions per second with a latency of tens of microseconds, crucially depend on fast distributed locking to mediate concurrent access to data partitioned in multiple servers. Unfortunately, acquiring locks becomes a significant bottleneck which severely limits the transaction throughput [7]. This is because servers have to spend their resources on (i) processing locking requests and (ii) aborting transactions that cannot acquire all locks under high-contention workloads, which can be otherwise used to execute and commit transactions. This is one of the main factors that led to relaxing consistency semantics in many recent large-scale distributed systems [8, 9], and the recent efforts to avoid coordination by leveraging application semantics [10, 11]. While these systems are successful in achieving high throughput, unfortunately, they restrict the programming model and complicate the

USENIX NSDI 2018

NetChain design goals

High throughput

Low latency

Already satisfied with the high-performance switches

Consistency

Fault tolerance

How to?

NetChain design goals

High throughput

Low latency

Consistency

Fault tolerance

Already satisfied with the high-performance switches

Chain replication in the network

Chain replication

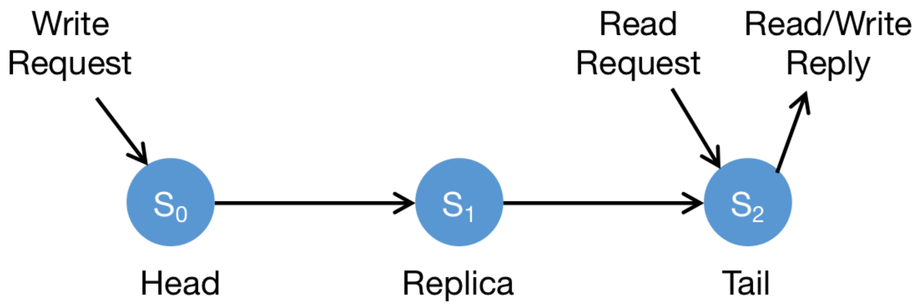
Storage nodes are organized in a chain structure

Handle operations:

- Read from the tail
- Write from head to tail

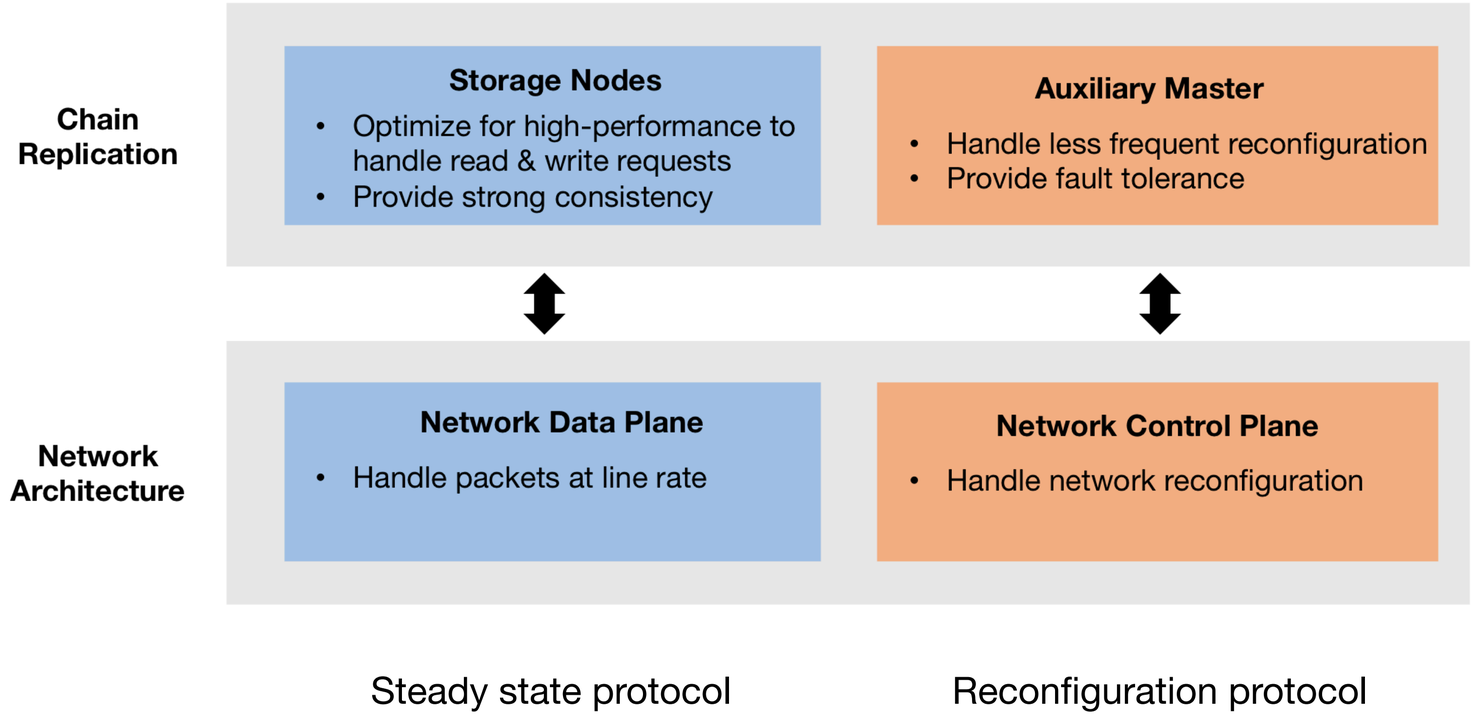
Provide strong consistency and fault tolerance

- Tolerate f failures with f+1 nodes

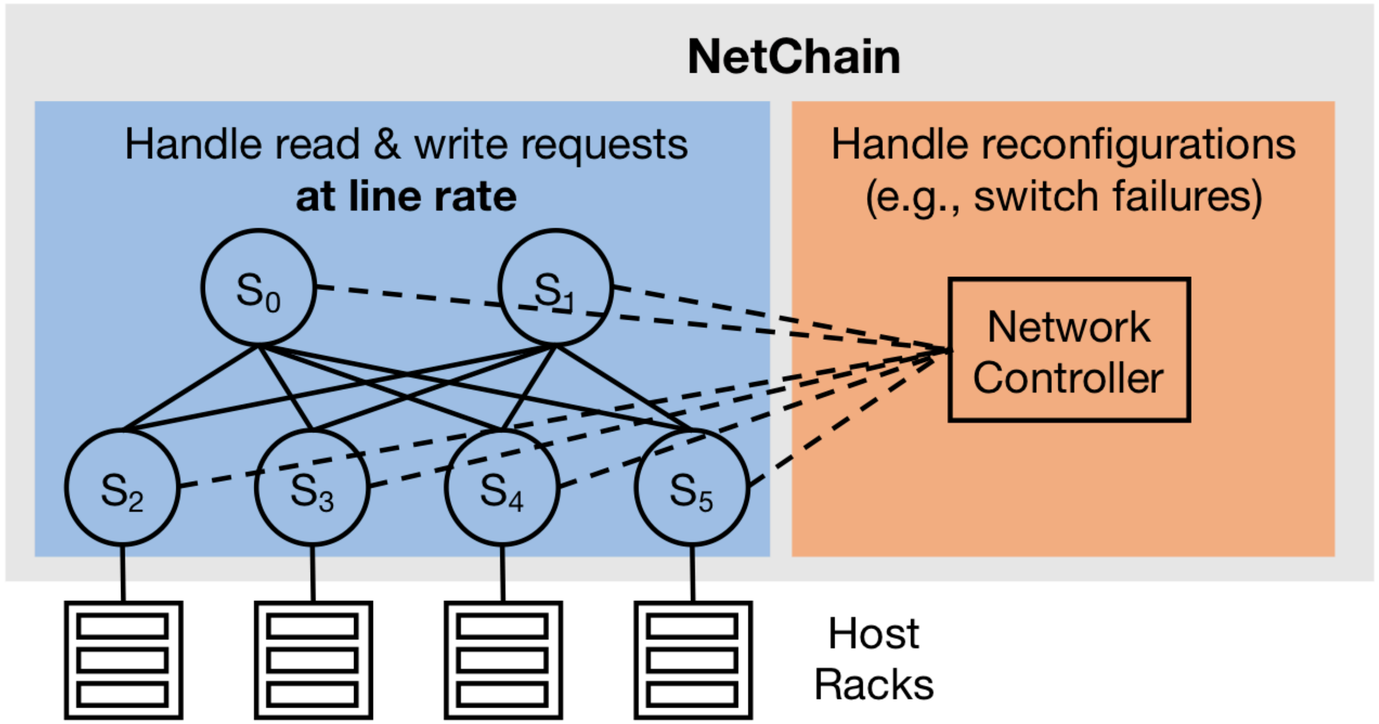


NetChain division of labor

Use Vertical Paxos to match the workload to network control and data planes



NetChain overview



NetChain challenges

Data plane

Control plane

How to store and serve key-value items?

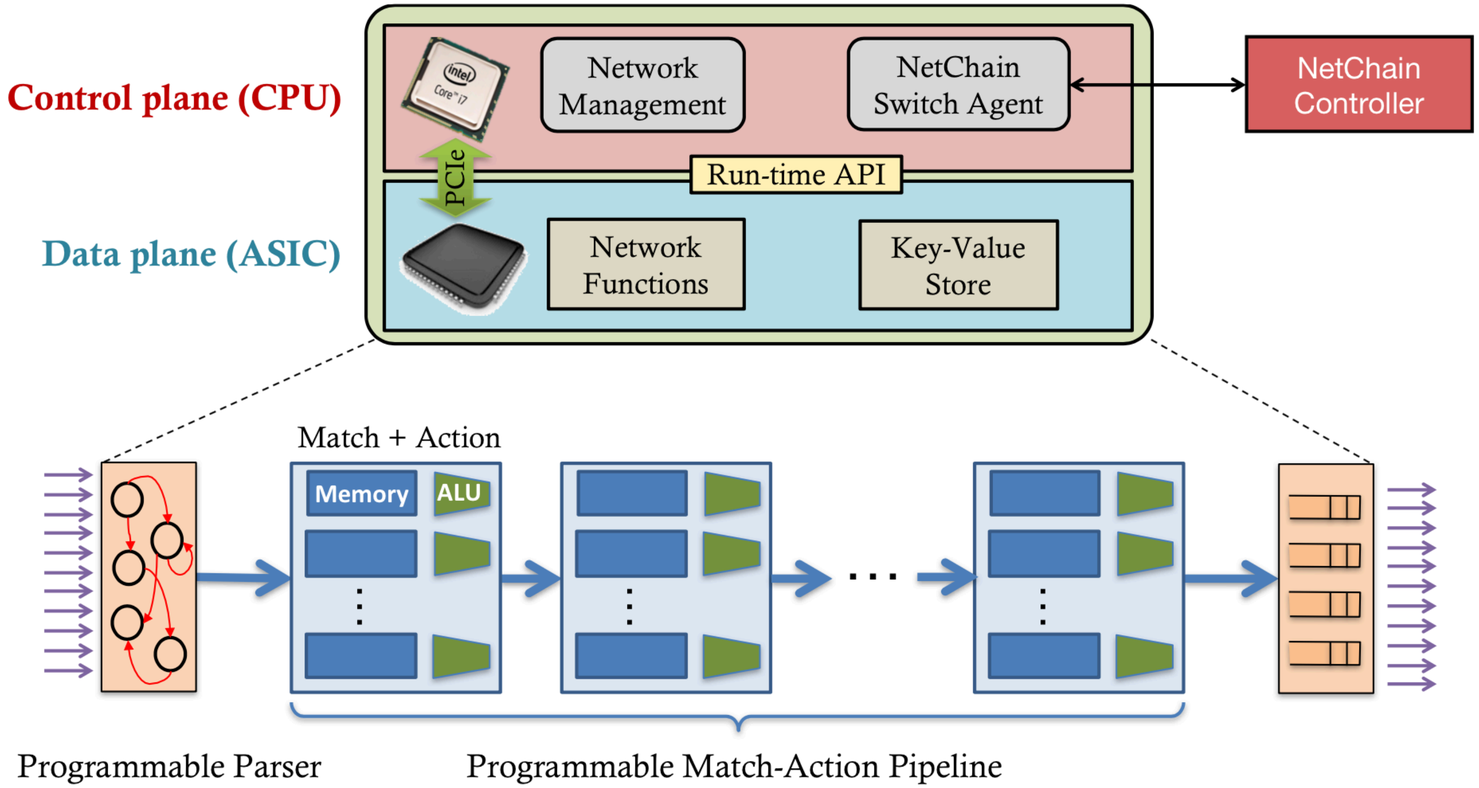
How to route queries according to the chain structure?

How to handle out of order delivery in the network?

How to handle switch failures?

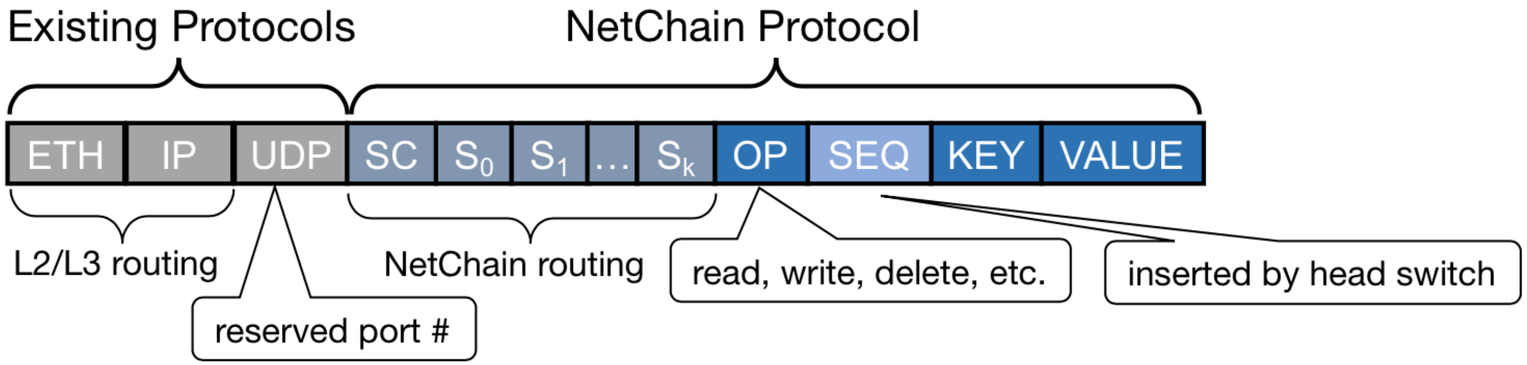
NetChain switch design

Similar to NetCache, expect the coordination components



NetChain packet format

Application-layer protocol: compatible with existing L2-L4 layers, invoked with a reserved UDP port



UDP is not reliable: upon packet loss, retry! Designing a reliable transport protocol for in-network computing is still an open challenge!

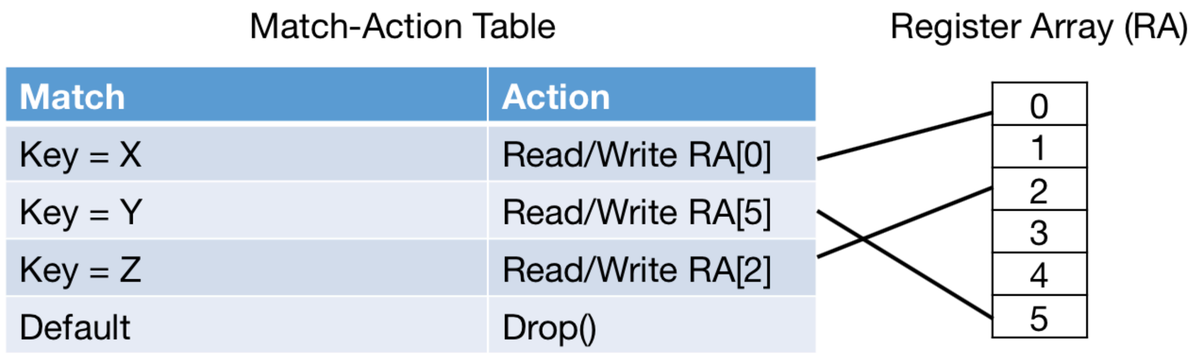
In-network key-value storage

Key-value store in a single switch

- Store and serve key-value items using register arrays

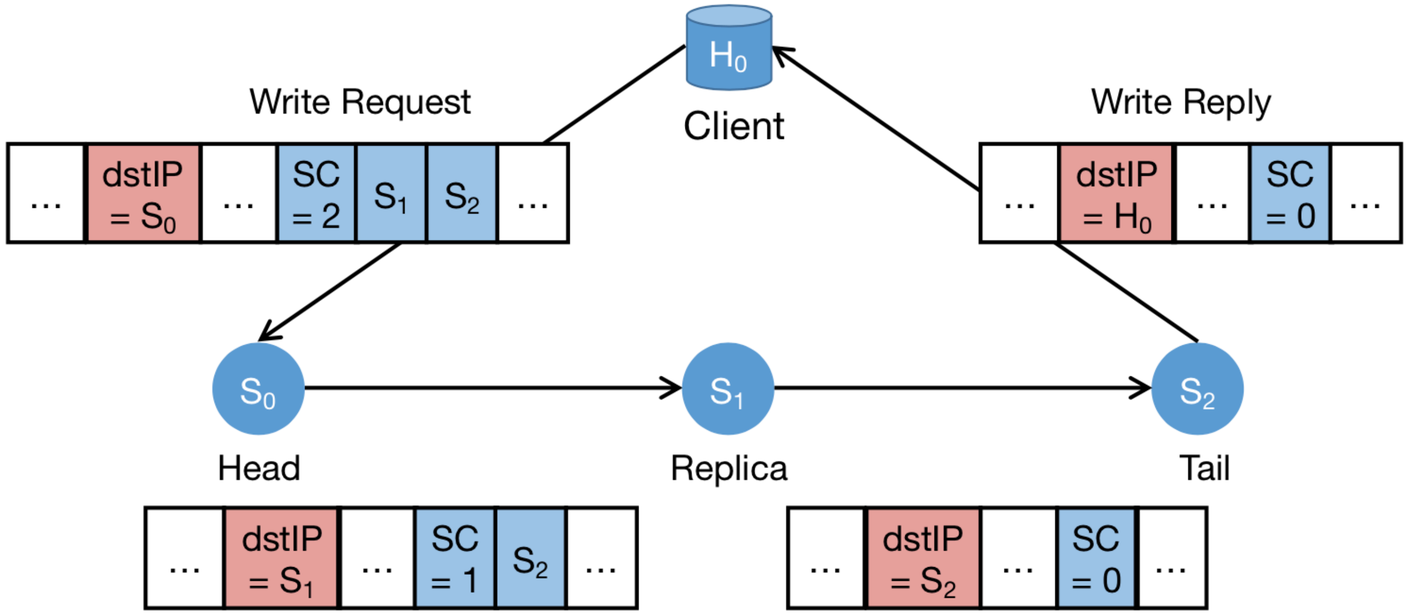
Key-value store in the network

- Data partitioning with consistent hashing and virtual nodes



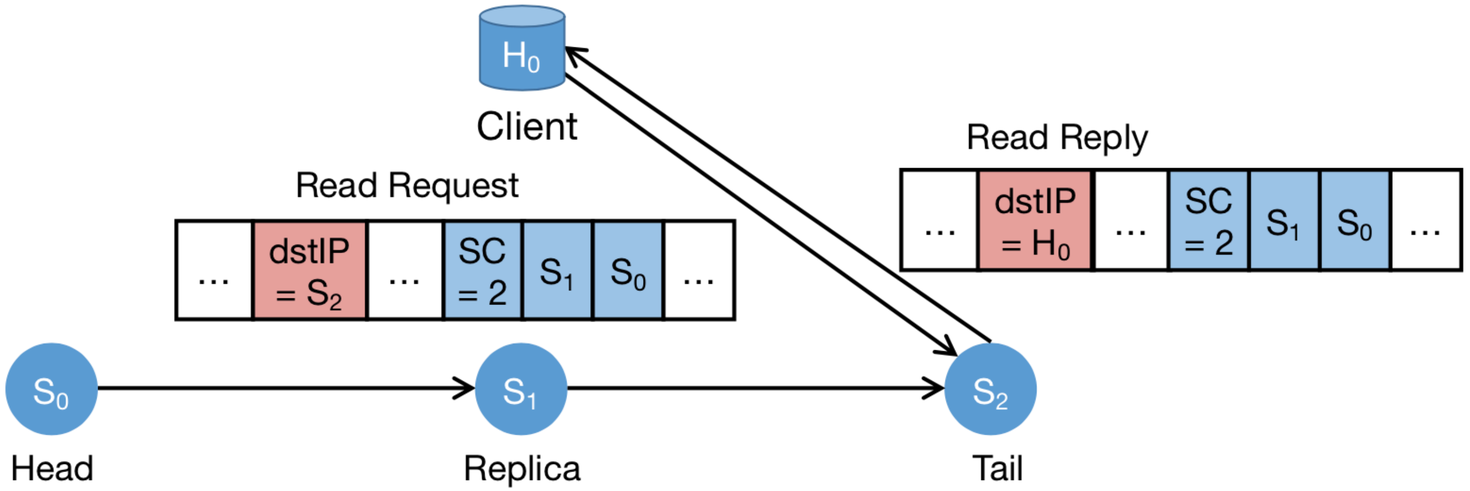
NetChain routing - write requests

Segment routing according to the chain structure

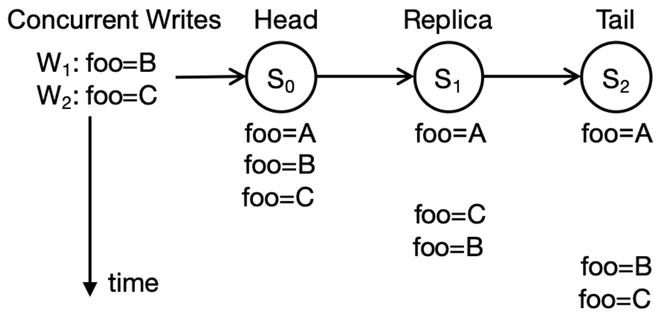


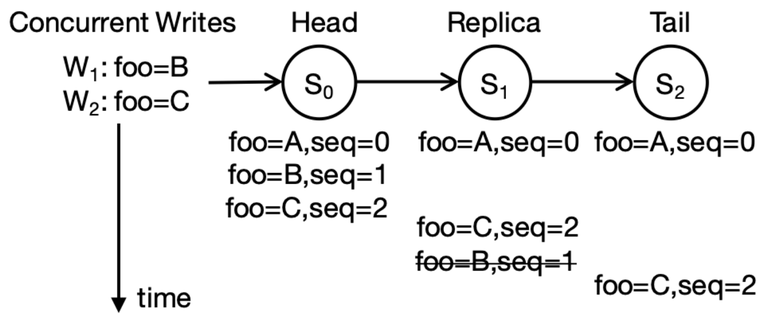
NetChain routing - read requests

Segment routing according to the chain structure



NetChain out of order delivery

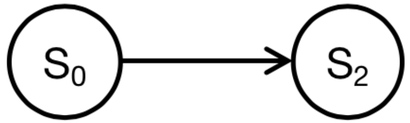




Serialization with sequence number!

Handling switch failures

Fast failover



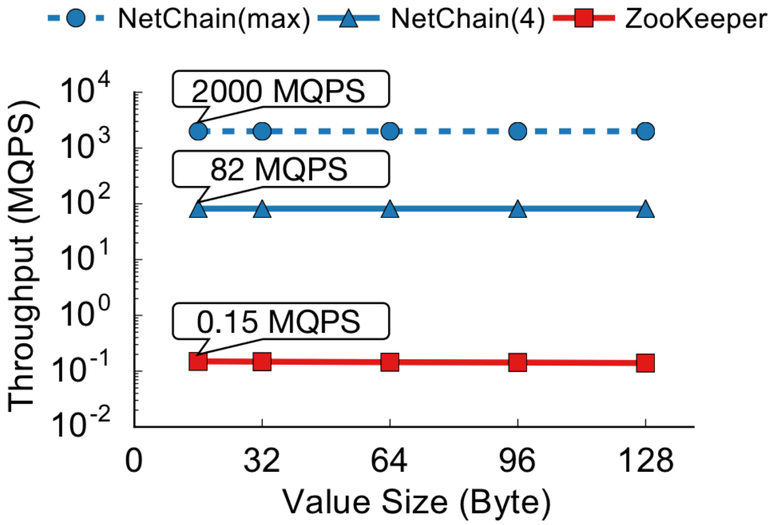
- Failover to remaining f nodes
- Tolerate f-1 failures
- Efficiency: only need to update neighbor switches of failed switch

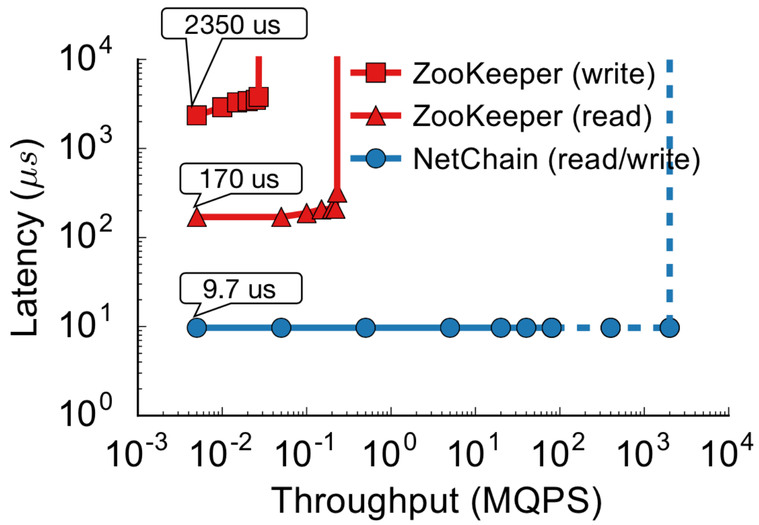
Failure recovery

- Add another switch
- Tolerate f+1 failures again
- Consistency: two-phase atomic switching
- Minimize disruption: virtual groups

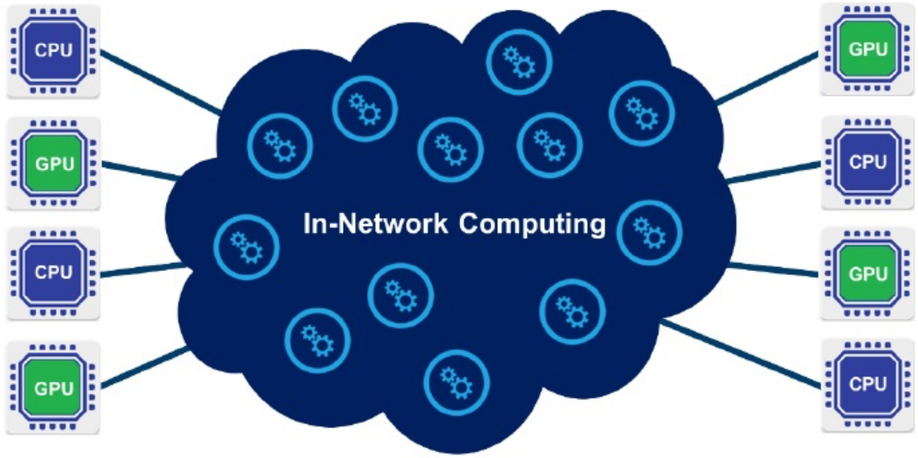
NetChain performance

Significant improvement over server-based solutions





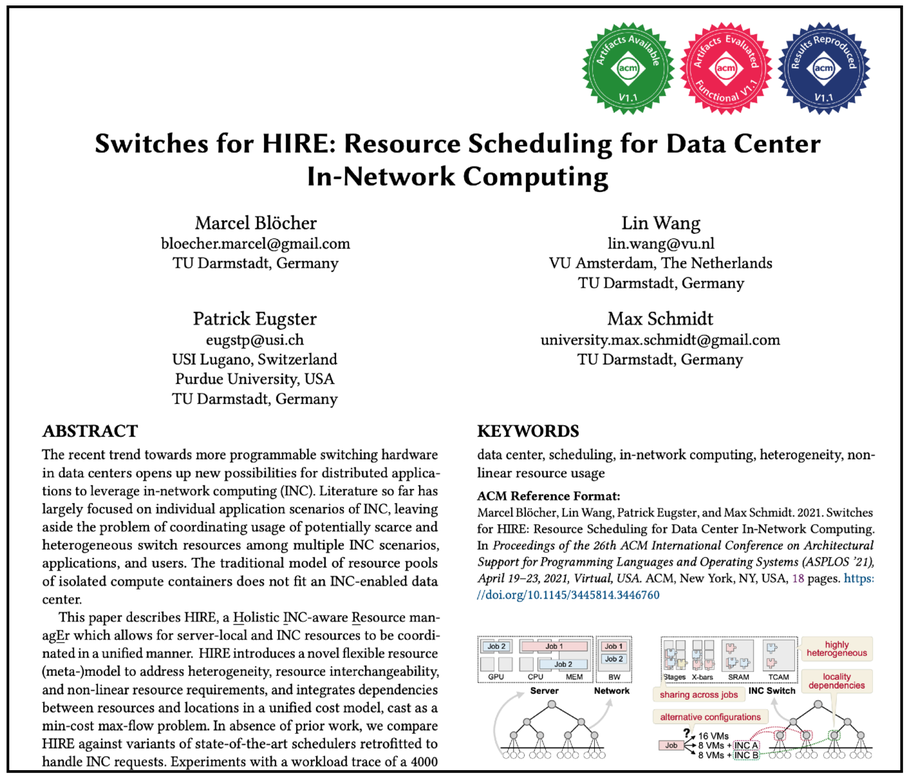
Summary



Leverage switches for in-network computing: in-network caching, in-network coordination

Ongoing research projects

Contact us if you are interested



Switches for HIRE: Resource Scheduling for Data Center In-Network Computing

Marcel Blöcher
bloecher.marcel@gmail.com
TU Darmstadt, Germany

Lin Wang
lin.wang@vu.nl
VU Amsterdam, The Netherlands
TU Darmstadt, Germany

Patrick Eugster
eugstp@usi.ch
USI Lugano, Switzerland
Purdue University, USA
TU Darmstadt, Germany

Max Schmidt
university.max.schmidt@gmail.com
TU Darmstadt, Germany

ABSTRACT

The recent trend towards more programmable switching hardware in data centers opens up new possibilities for distributed applications to leverage in-network computing (INC). Literature so far has largely focused on individual application scenarios of INC, leaving aside the problem of coordinating usage of potentially scarce and heterogeneous switch resources among multiple INC scenarios, applications, and users. The traditional model of resource pools of isolated compute containers does not fit an INC-enabled data center.

This paper describes HIRE, a Holistic INC-aware Resource managEr which allows for server-local and INC resources to be coordinated in a unified manner. HIRE introduces a novel flexible resource (meta-)model to address heterogeneity, resource interchangeability, and non-linear resource requirements, and integrates dependencies between resources and locations in a unified cost model, cast as a min-cost max-flow problem. In absence of prior work, we compare HIRE against variants of state-of-the-art schedulers retrofitted to handle INC requests. Experiments with a workload trace of a 4000

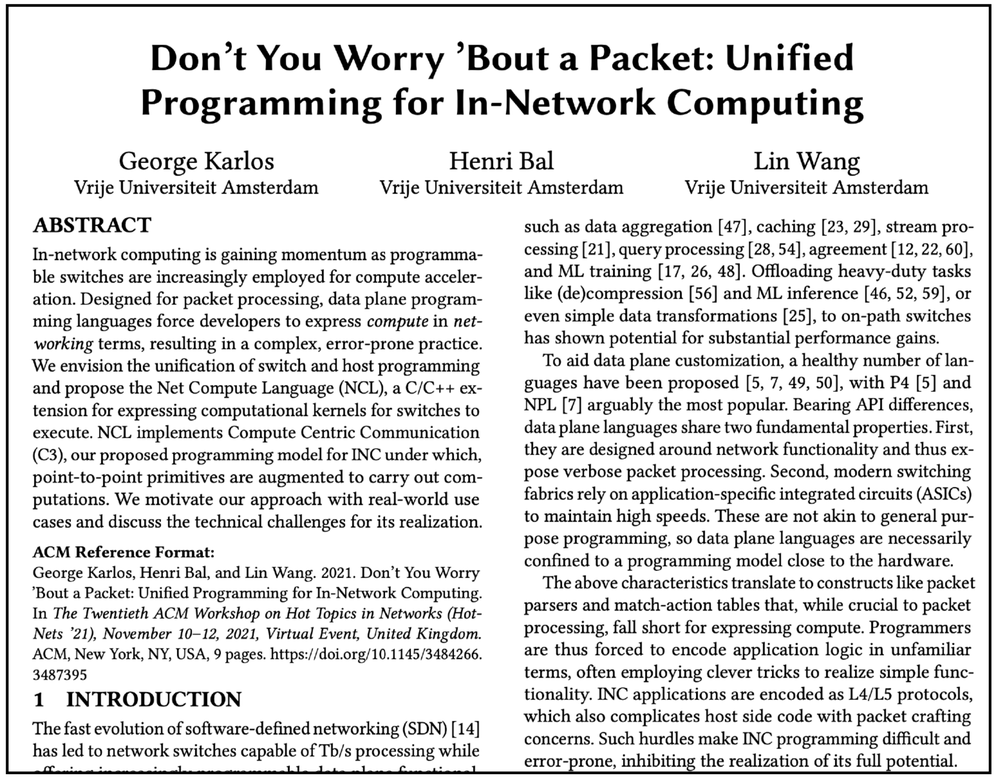
KEYWORDS

data center, scheduling, in-network computing, heterogeneity, non-linear resource usage

ACM Reference Format:

Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: Resource Scheduling for Data Center In-Network Computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3445814.3446760

ACM ASPLOS 2021



Don't You Worry 'Bout a Packet: Unified Programming for In-Network Computing

George Karlos
Vrije Universiteit Amsterdam

Henri Bal
Vrije Universiteit Amsterdam

Lin Wang
Vrije Universiteit Amsterdam

ABSTRACT

In-network computing is gaining momentum as programmable switches are increasingly employed for compute acceleration. Designed for packet processing, data plane programming languages force developers to express *compute* in *networking* terms, resulting in a complex, error-prone practice. We envision the unification of switch and host programming and propose the Net Compute Language (NCL), a C/C++ extension for expressing computational kernels for switches to execute. NCL implements Compute Centric Communication (C3), our proposed programming model for INC under which, point-to-point primitives are augmented to carry out computations. We motivate our approach with real-world use cases and discuss the technical challenges for its realization.

ACM Reference Format:

George Karlos, Henri Bal, and Lin Wang. 2021. Don't You Worry 'Bout a Packet: Unified Programming for In-Network Computing. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3484266.3487395

1 INTRODUCTION

The fast evolution of software-defined networking (SDN) [14] has led to network switches capable of Tb/s processing while

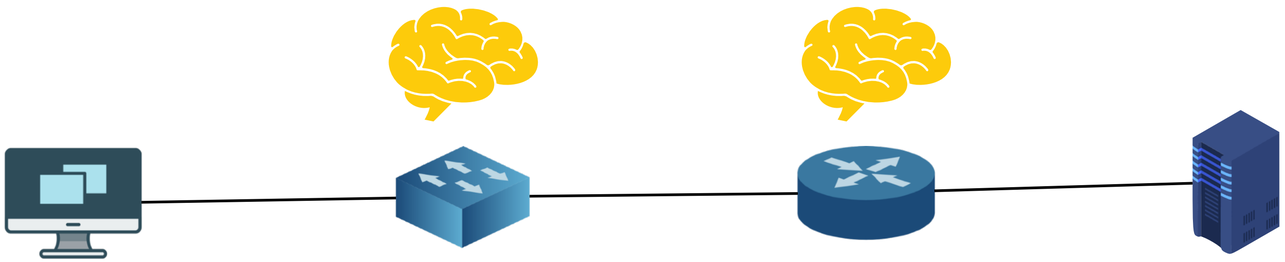
such as data aggregation [47], caching [23, 29], stream processing [21], query processing [28, 54], agreement [12, 22, 60], and ML training [17, 26, 48]. Offloading heavy-duty tasks like (de)compression [56] and ML inference [46, 52, 59], or even simple data transformations [25], to on-path switches has shown potential for substantial performance gains.

To aid data plane customization, a healthy number of languages have been proposed [5, 7, 49, 50], with P4 [5] and NPL [7] arguably the most popular. Bearing API differences, data plane languages share two fundamental properties. First, they are designed around network functionality and thus expose verbose packet processing. Second, modern switching fabrics rely on application-specific integrated circuits (ASICs) to maintain high speeds. These are not akin to general purpose programming, so data plane languages are necessarily confined to a programming model close to the hardware.

The above characteristics translate to constructs like packet parsers and match-action tables that, while crucial to packet processing, fall short for expressing compute. Programmers are thus forced to encode application logic in unfamiliar terms, often employing clever tricks to realize simple functionality. INC applications are encoded as L4/L5 protocols, which also complicates host side code with packet crafting concerns. Such hurdles make INC programming difficult and error-prone, inhibiting the realization of its full potential.

ACM HotNets 2021

Next time: machine learning for networking



Can we use machine learning to power the intelligence of the network?