Advanced Topics in Communication Networks

Programming Network Data Planes

Laurent Vanbever

nsg.ee.ethz.ch

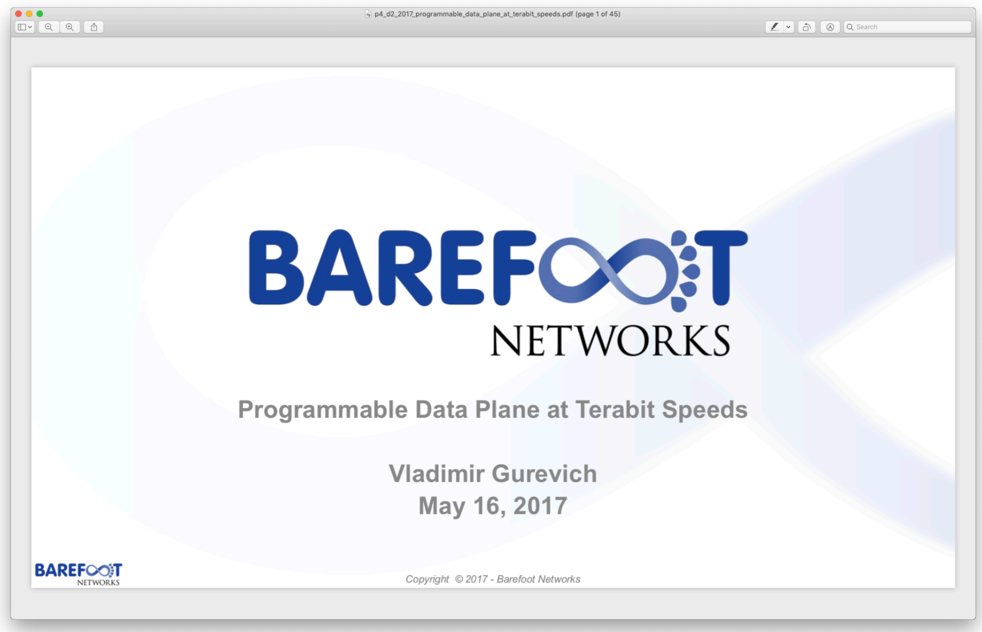
ETH Zürich

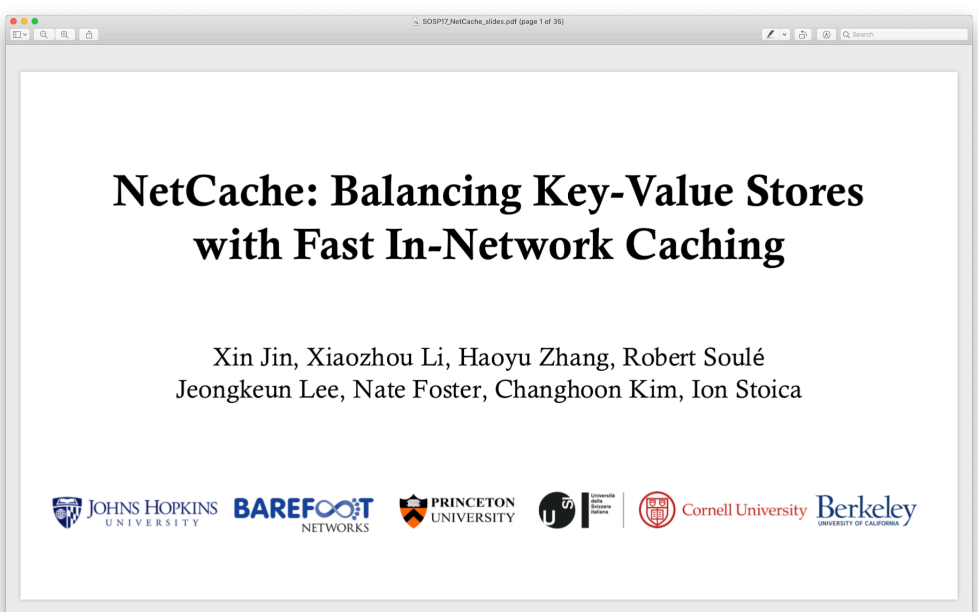
Nov 1 2018

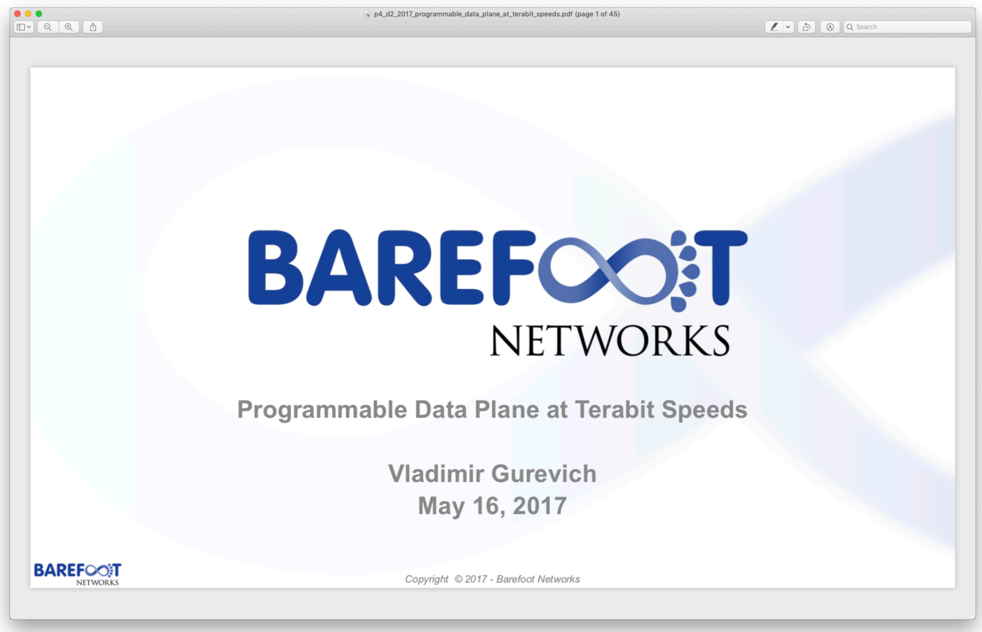
Last week on

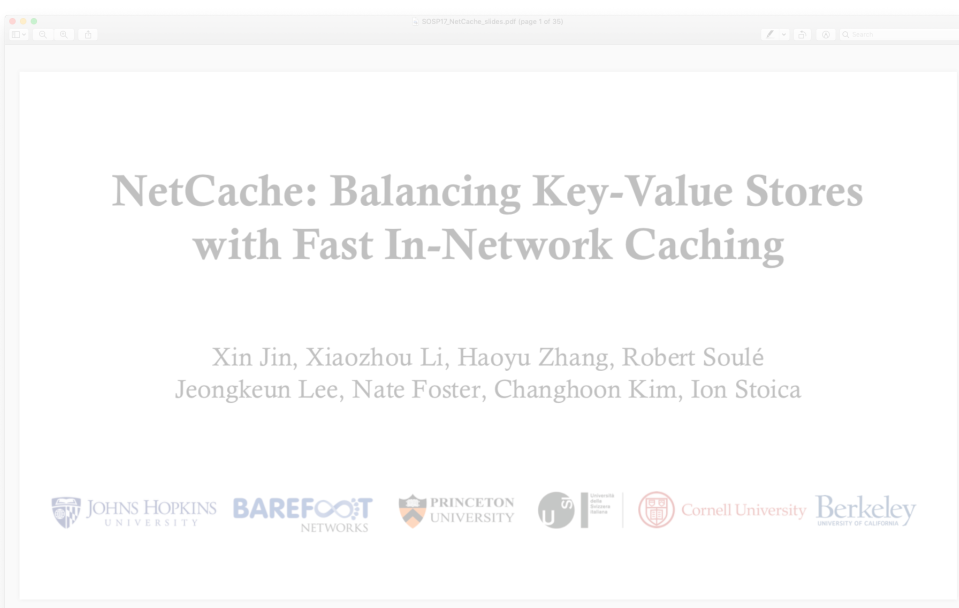
Advanced Topics in Communication Networks

We looked at the Tofino architecture together with two (key, value) store applications: Net/{Cache, Chain}






BAREFOOT
NETWORKS
Programmable Data Plane at Terabit Speeds
Vladimir Gurevich
May 16, 2017
BAREFOOT NETWORKS
Copyright © 2017 - Barefoot Networks


NetCache: Balancing Key-Value Stores with Fast In-Network Caching
Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé
Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica
JOHNS HOPKINS UNIVERSITY
BAREFOOT NETWORKS
PRINCETON UNIVERSITY
Cornell University
Berkeley


NetChain: Scale-Free Sub-RTT Coordination
Xin Jin
Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee,
Nate Foster, Changhoon Kim, Ion Stoica
JOHNS HOPKINS UNIVERSITY
BAREFOOT NETWORKS
PRINCETON UNIVERSITY
Berkeley

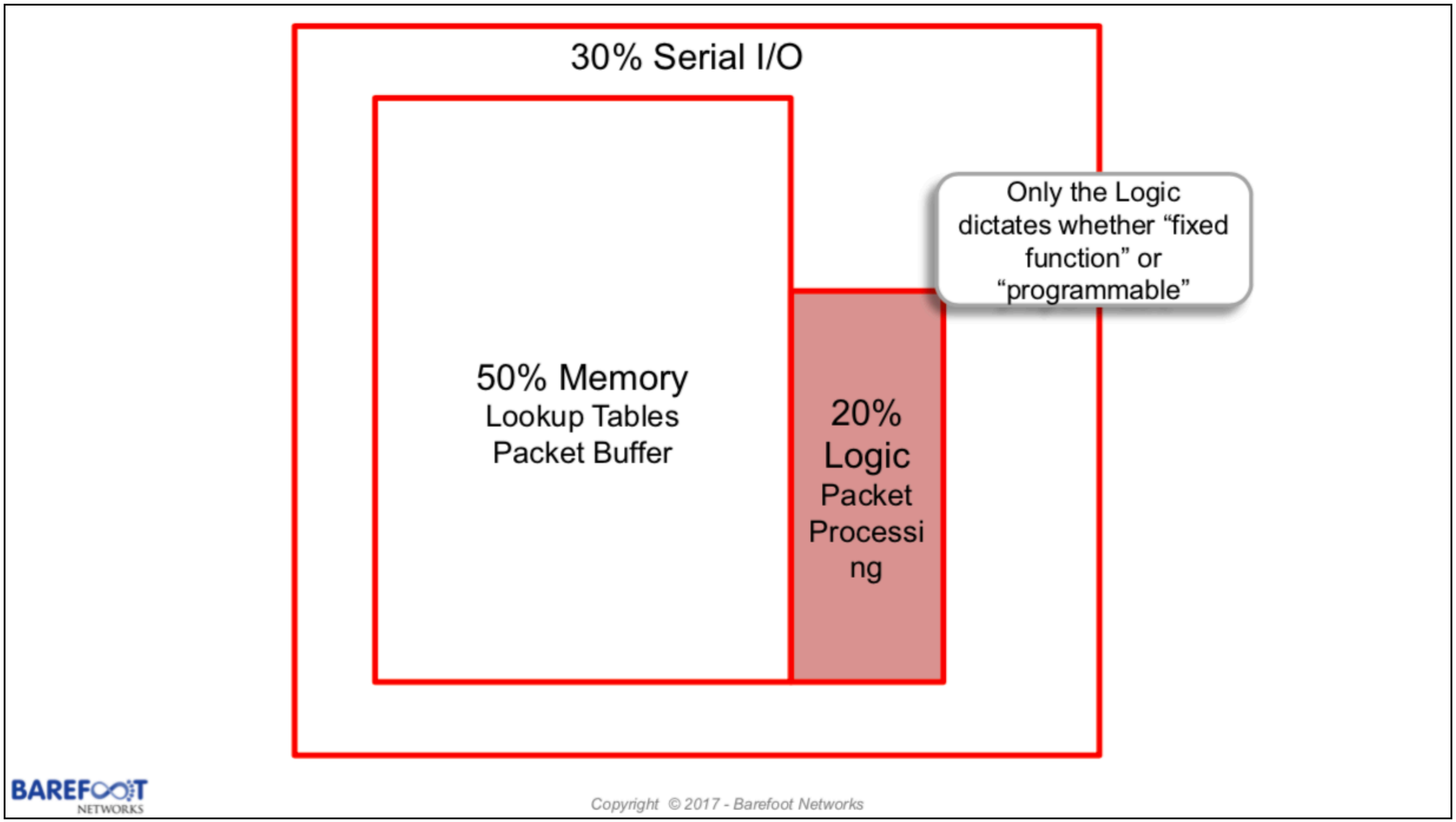
“Programmable switches are 10-100x slower than fixed-function switches. They cost more and consume more power.”

Conventional wisdom in networking

BAREFOOT NETWORKS

Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

One of the main enabler for data-plane programmability is the shrinking size of the packet processing logic chip.



Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

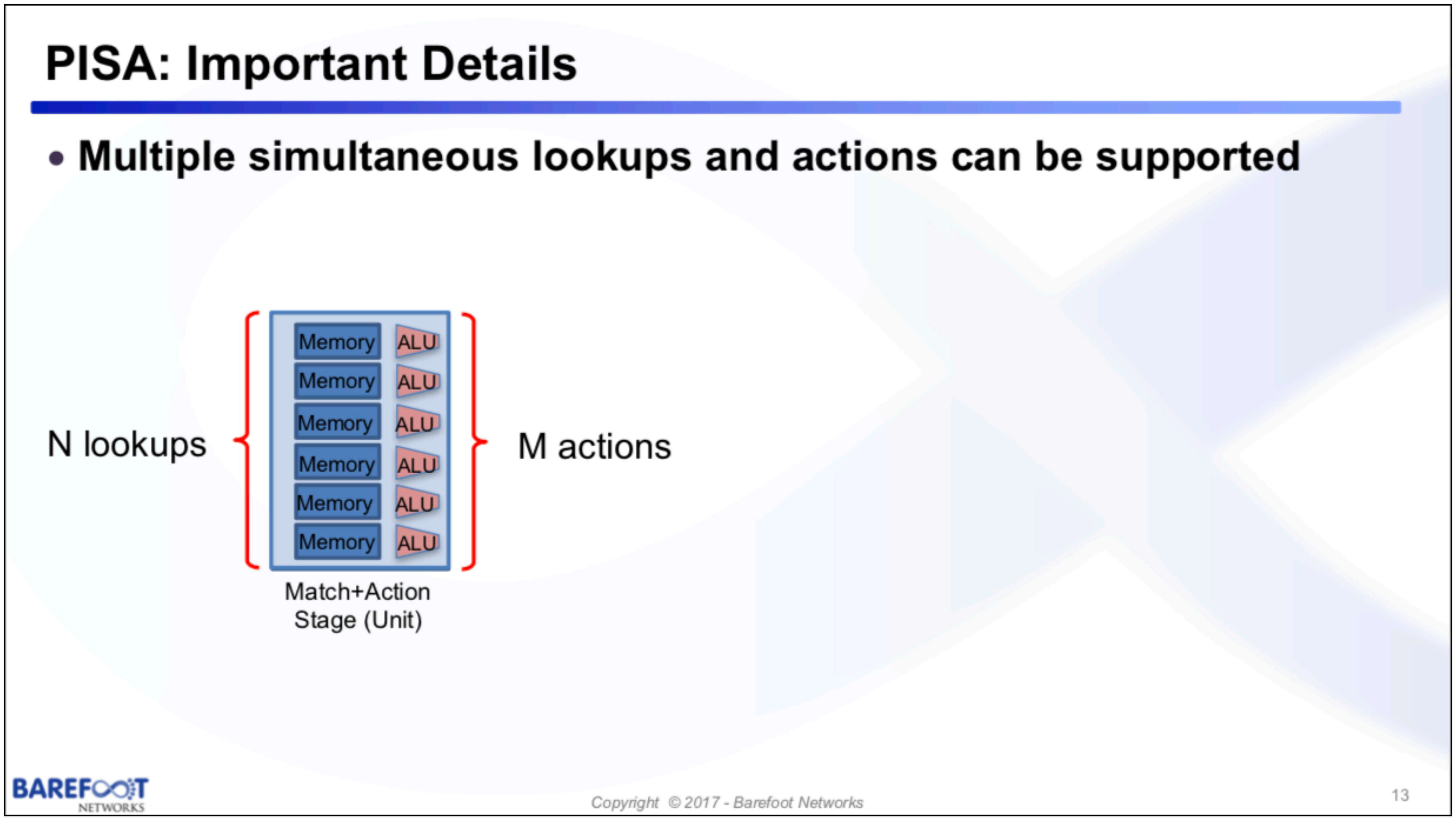
Barefoot Tofino processes packets in parallel,

even though the semantic of a P4 program is sequential

Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Barefoot Tofino processes packets in parallel,

even though the semantic of a P4 program is sequential



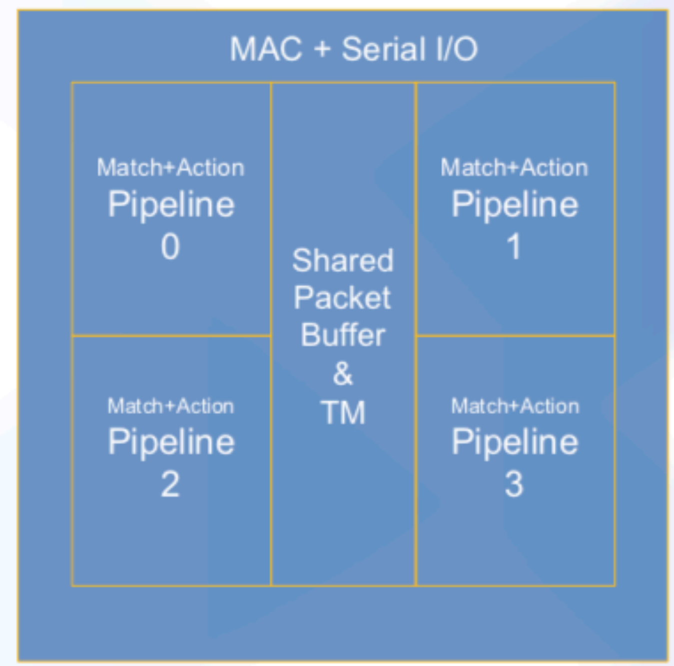
Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Barefoot Tofino 6.5 Tbps backplane

several billion packets per second at line rate

6.5Tb/s Tofino™ Summary

- **State of the art design**
  - Single Shared Packet Buffer
  - TSMC 16nm FinFET+
- **Four Match+Action Pipelines**
  - Fully programmable PISA Embodiment
  - All compiled programs run at line-rate.
  - Up to 1.3 million IPv4 routes
- **Port Configurations**
  - 65 x 100GE/40GE
  - 130 x 50GE
  - 260 x 25GE/10GE
- **CPU Interfaces**
  - PCIe: Gen3 x4/x2/x1
  - Dedicated 100GE port



BAREFOOT NETWORKS

Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Tofino relies on Packet Header Vector (PHV) to pass states between stages—this is one of the limiting factor

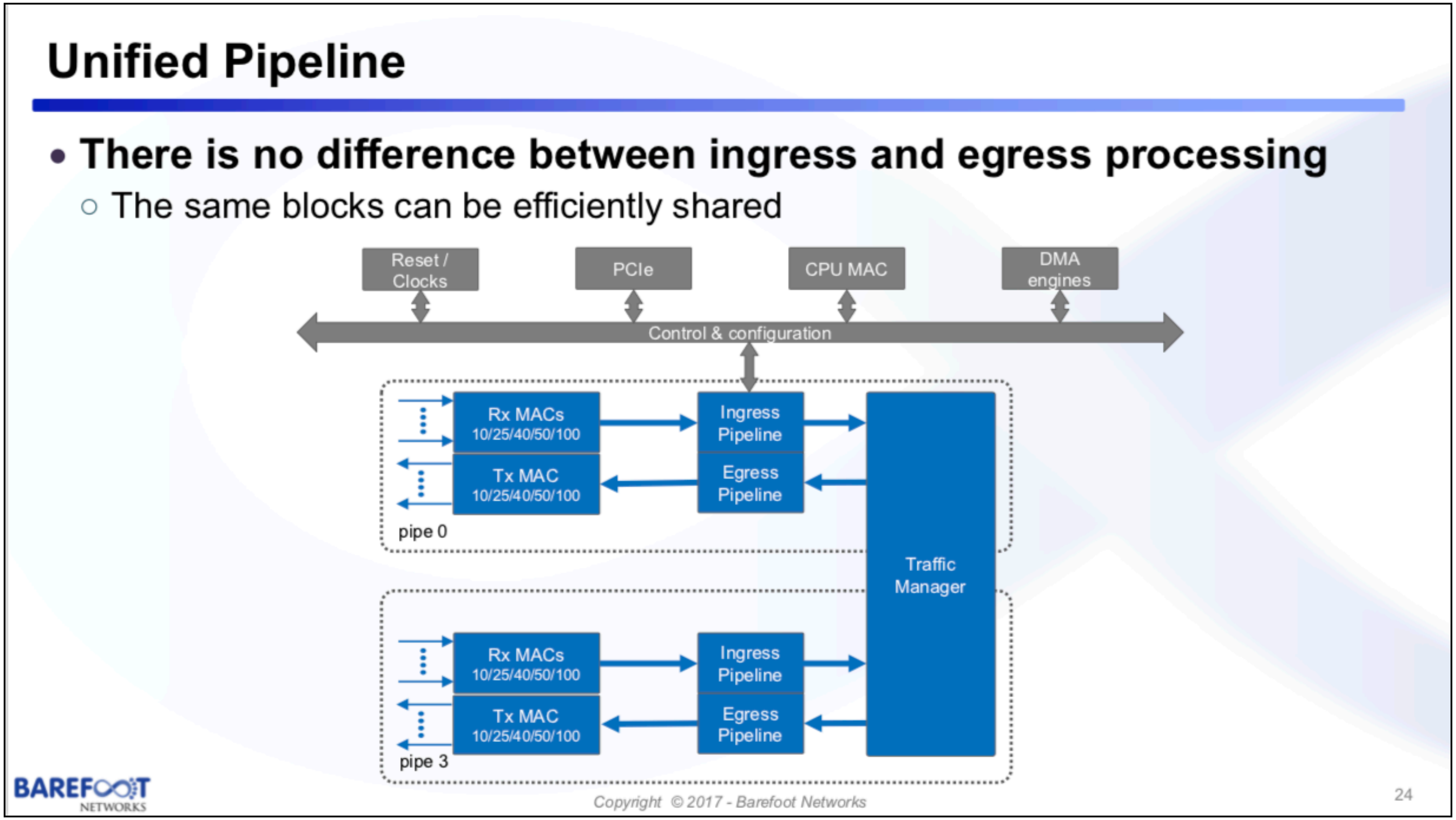
Packet Header Vector (PHV)

- A set of uniform containers that carry the headers and metadata along the pipeline
- Fields can be packed into any container or their combination
- PHV Allocation step in the compiler decides the actual packing

BAREFOOT NETWORKS

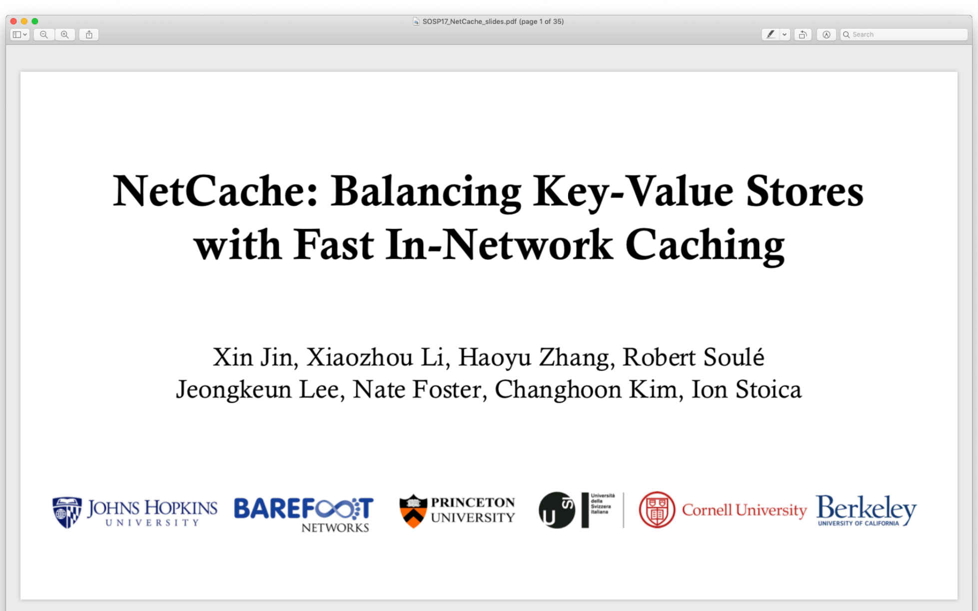
Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017

Tofino uses a folded pipeline in which the *same* stages are used for both the ingress and the egress pipeline



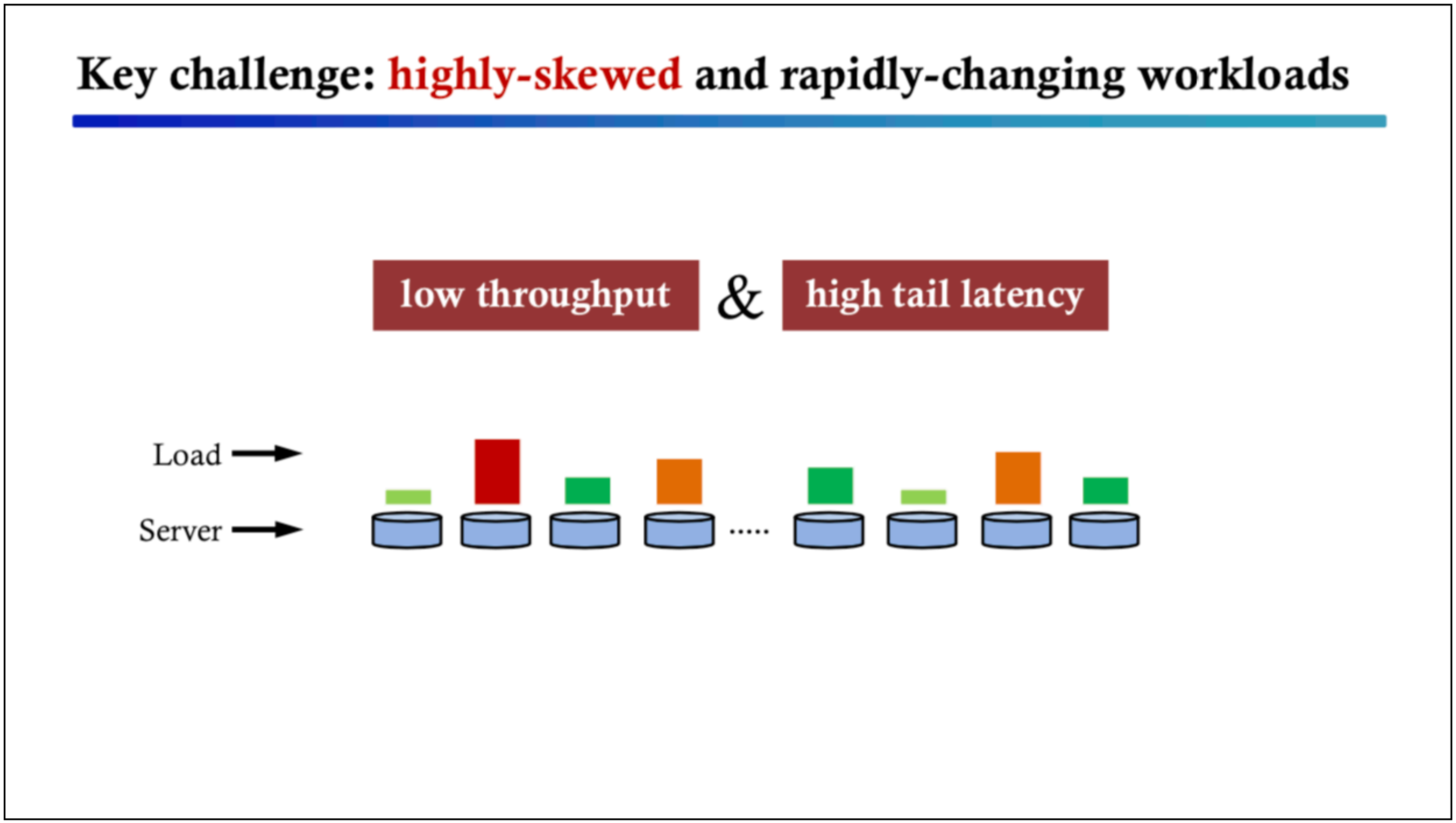
Source: Programmable Data Planes at Terabit Speeds, Vladimir Gurevich, 2017


BAREFOOT
NETWORKS
Programmable Data Plane at Terabit Speeds
Vladimir Gurevich
May 16, 2017
Copyright © 2017 - Barefoot Networks


NetCache: Balancing Key-Value Stores with Fast In-Network Caching
Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé
Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica
Johns Hopkins University
BAREFOOT NETWORKS
Princeton University
Cornell University
Berkeley University of California

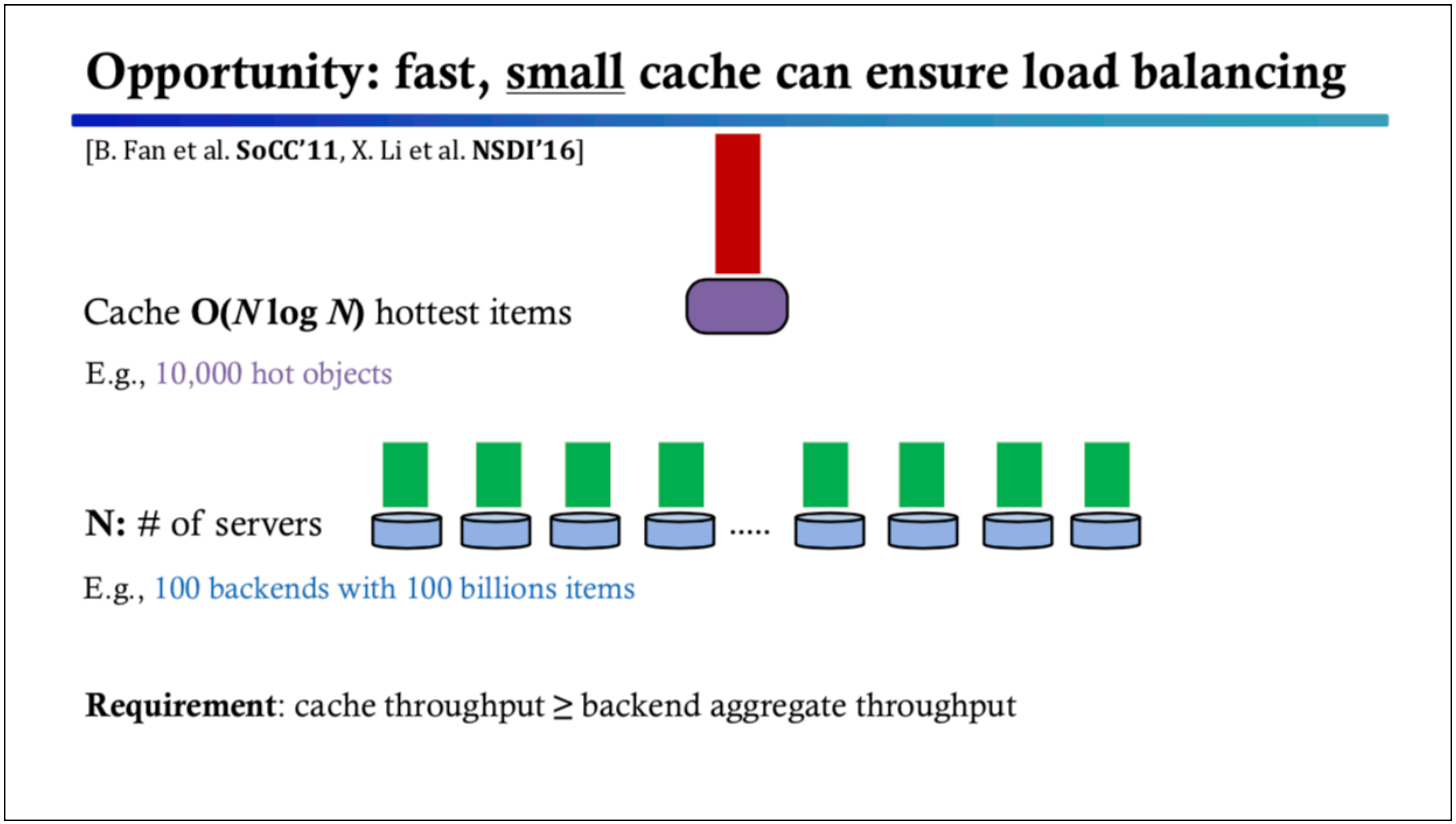

NetChain: Scale-Free Sub-RTT Coordination
Xin Jin
Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee,
Nate Foster, Changhoon Kim, Ion Stoica
Johns Hopkins University
BAREFOOT NETWORKS
Princeton University
Berkeley University of California

NetCache solves the problem of load-balancing in key-values stores observing *dynamic*, *skewed* workload



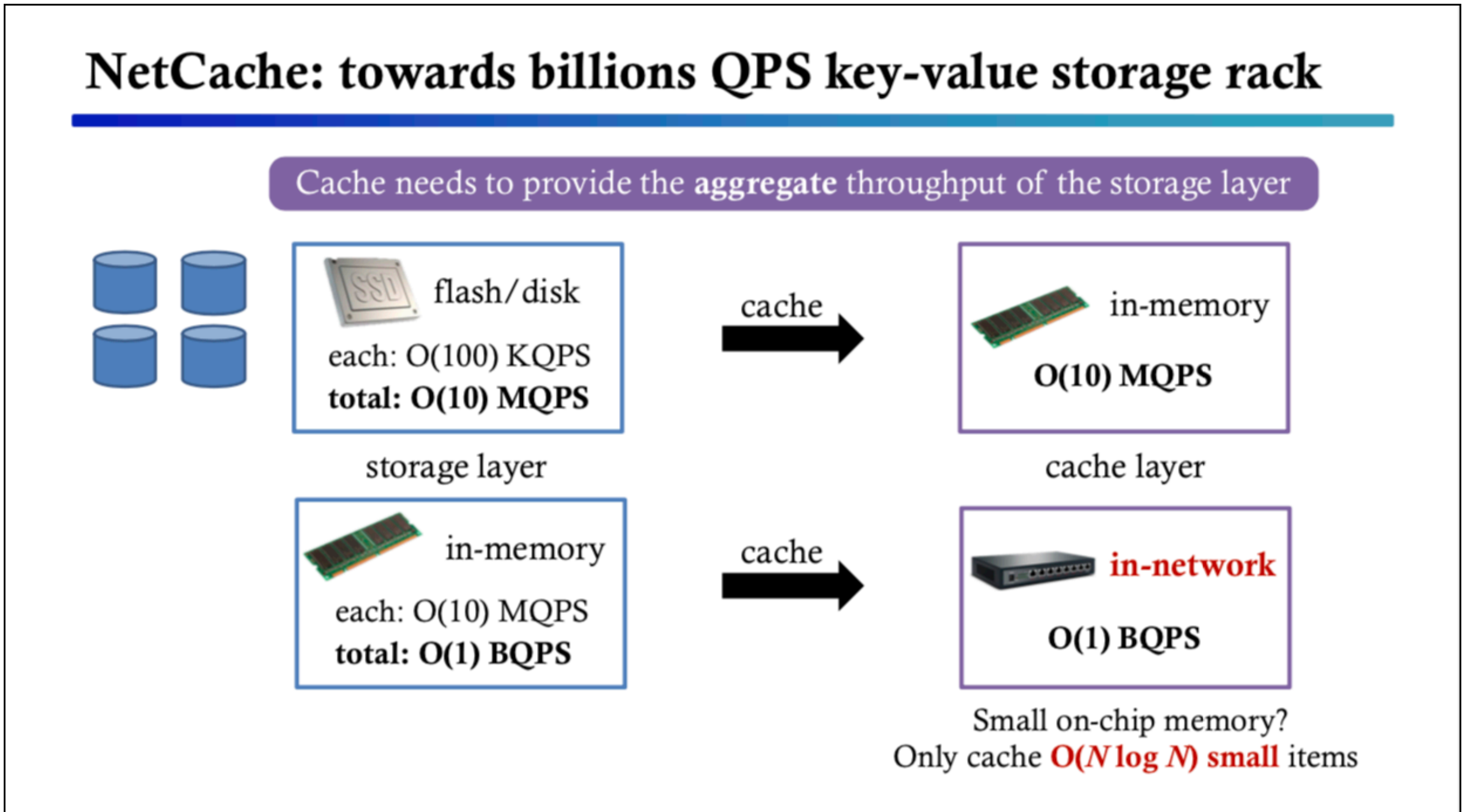
Source: NetCache: Balancing Key-Value Stores with Fast In-Network Caching, Xin Jin, 2017

It leverages that a small but very fast cache can provide perfect load-balancing… in theory



Source: NetCache: Balancing Key-Value Stores with Fast In-Network Caching, Xin Jin, 2017

NetCache relies on the O(billion) throughput of programmable network devices to achieve it in practice



Source: NetCache: Balancing Key-Value Stores with Fast In-Network Caching, Xin Jin, 2017

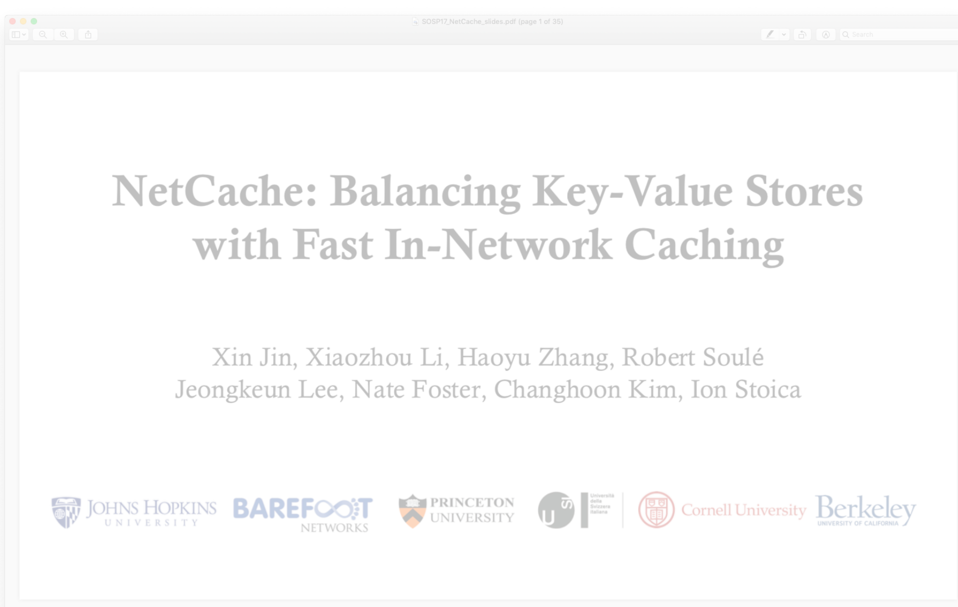
It relies on a tailored UDP-based protocol, an de/encoding scheme for storing variable length values, and sketches

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

- How to identify application-level packet fields ?
- How to store and serve variable-length data ?
- How to efficiently keep the cache up-to-date ?

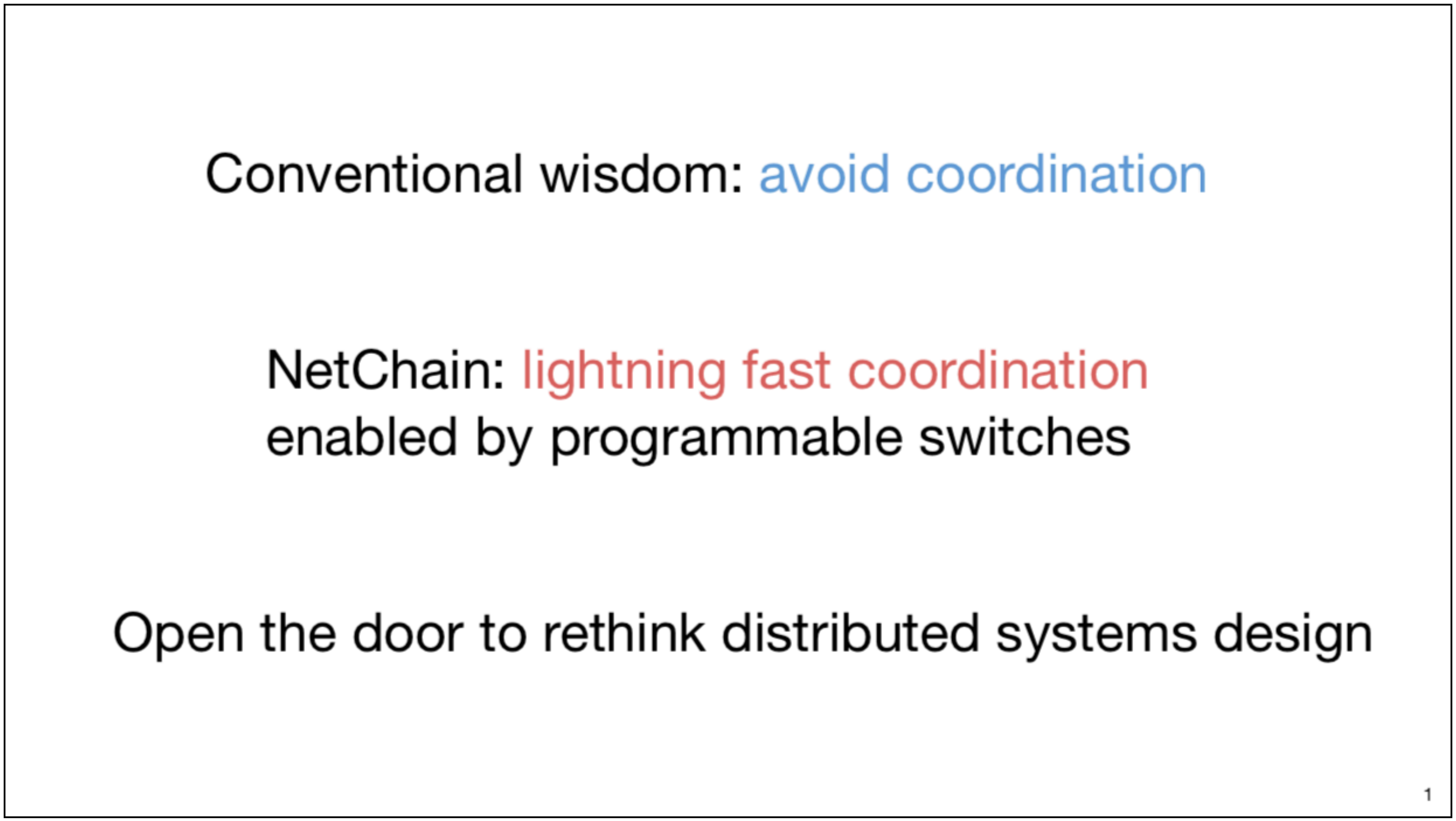
Source: NetCache: Balancing Key-Value Stores with Fast In-Network Caching, Xin Jin, 2017


BAREFOOT
NETWORKS
Programmable Data Plane at Terabit Speeds
Vladimir Gurevich
May 16, 2017
Copyright © 2017 - Barefoot Networks


NetCache: Balancing Key-Value Stores with Fast In-Network Caching
Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé
Jeongkeun Lee, Nate Foster, Changhoon Kim, Ion Stoica

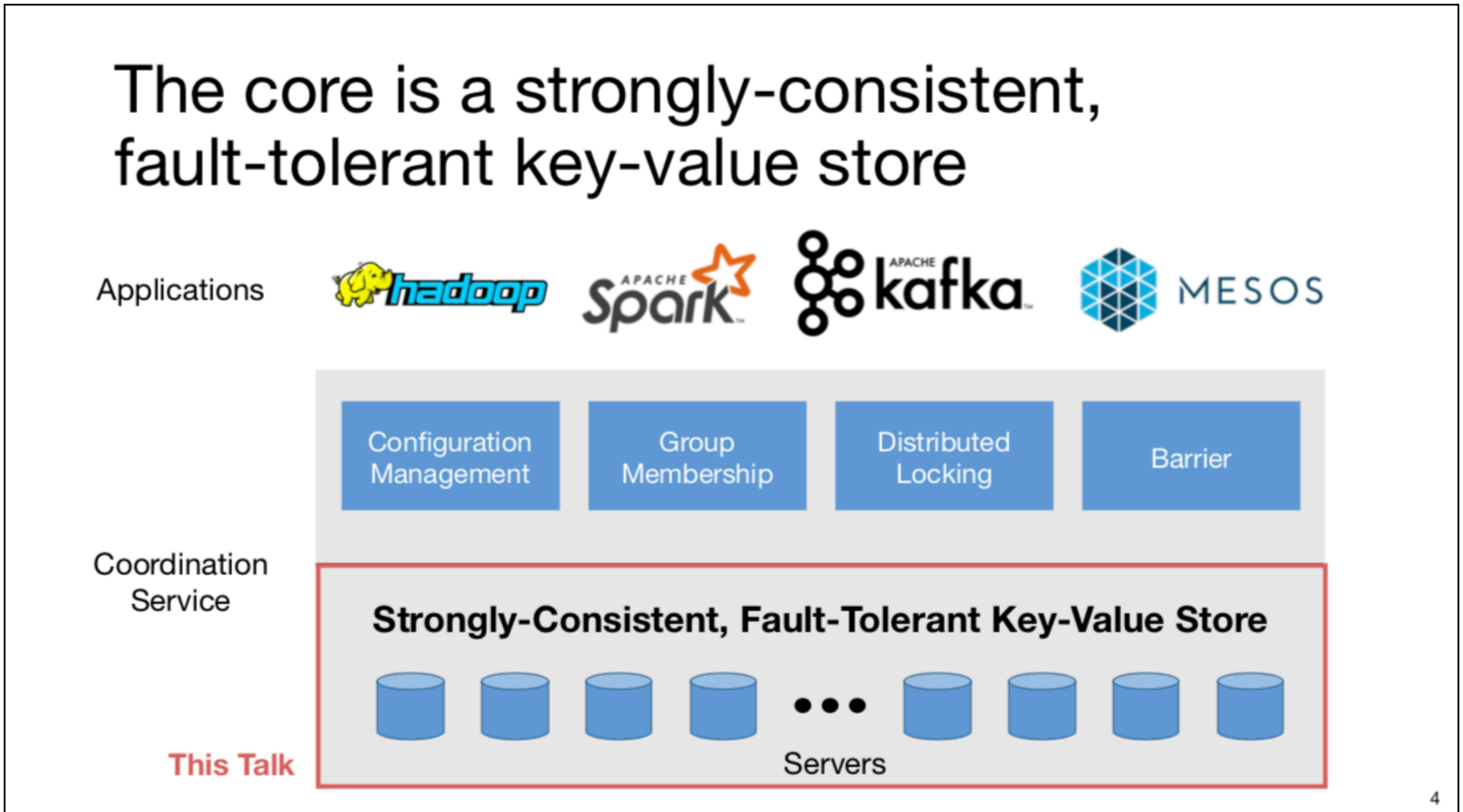

NetChain: Scale-Free Sub-RTT Coordination
Xin Jin
Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee,
Nate Foster, Changhoon Kim, Ion Stoica

NetChain builds upon NetCache to scale coordination services, a key building block of distributed systems



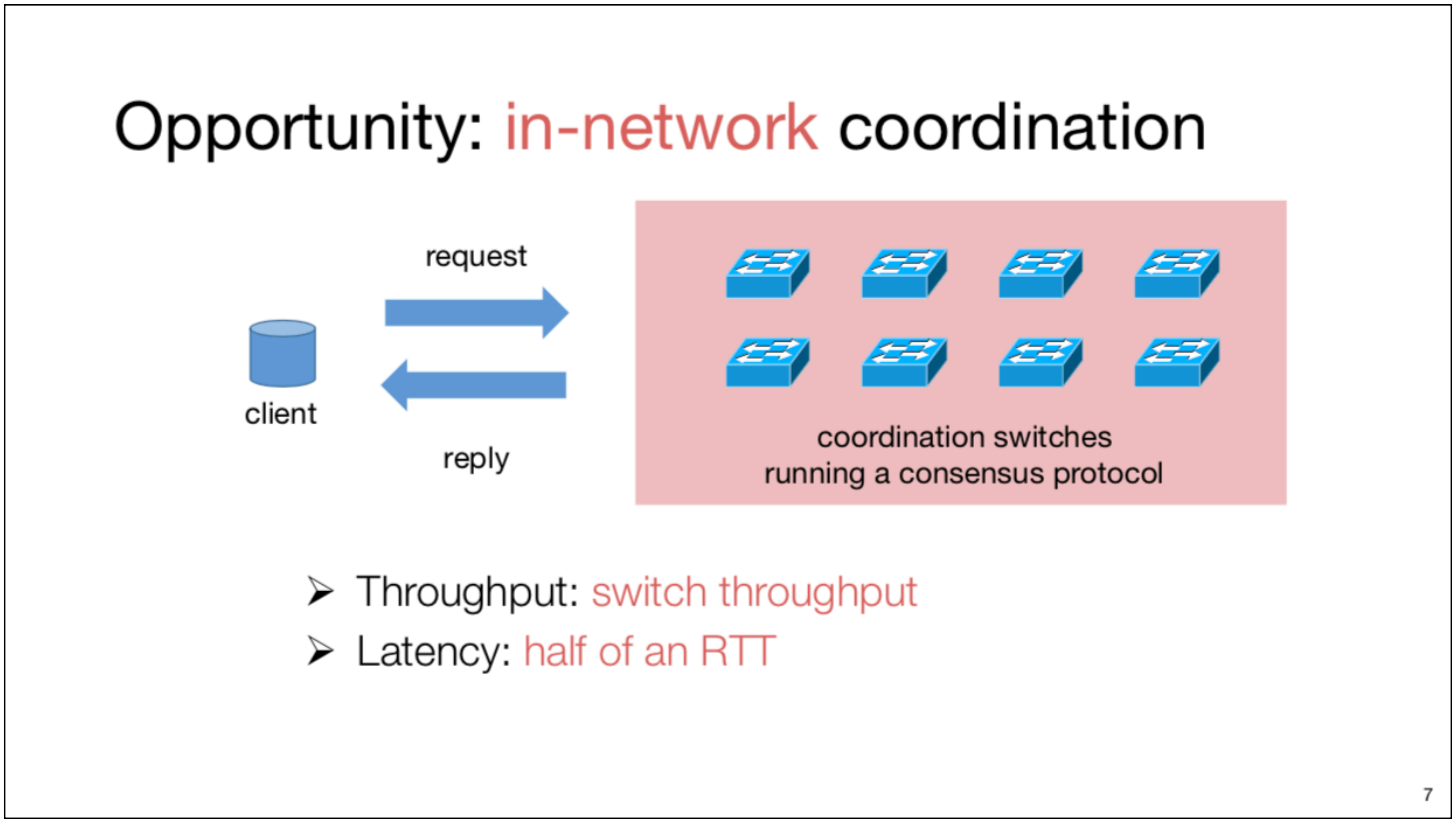
Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

Coordination services typically rely on a replicated key-value store for consistency and fault-tolerance



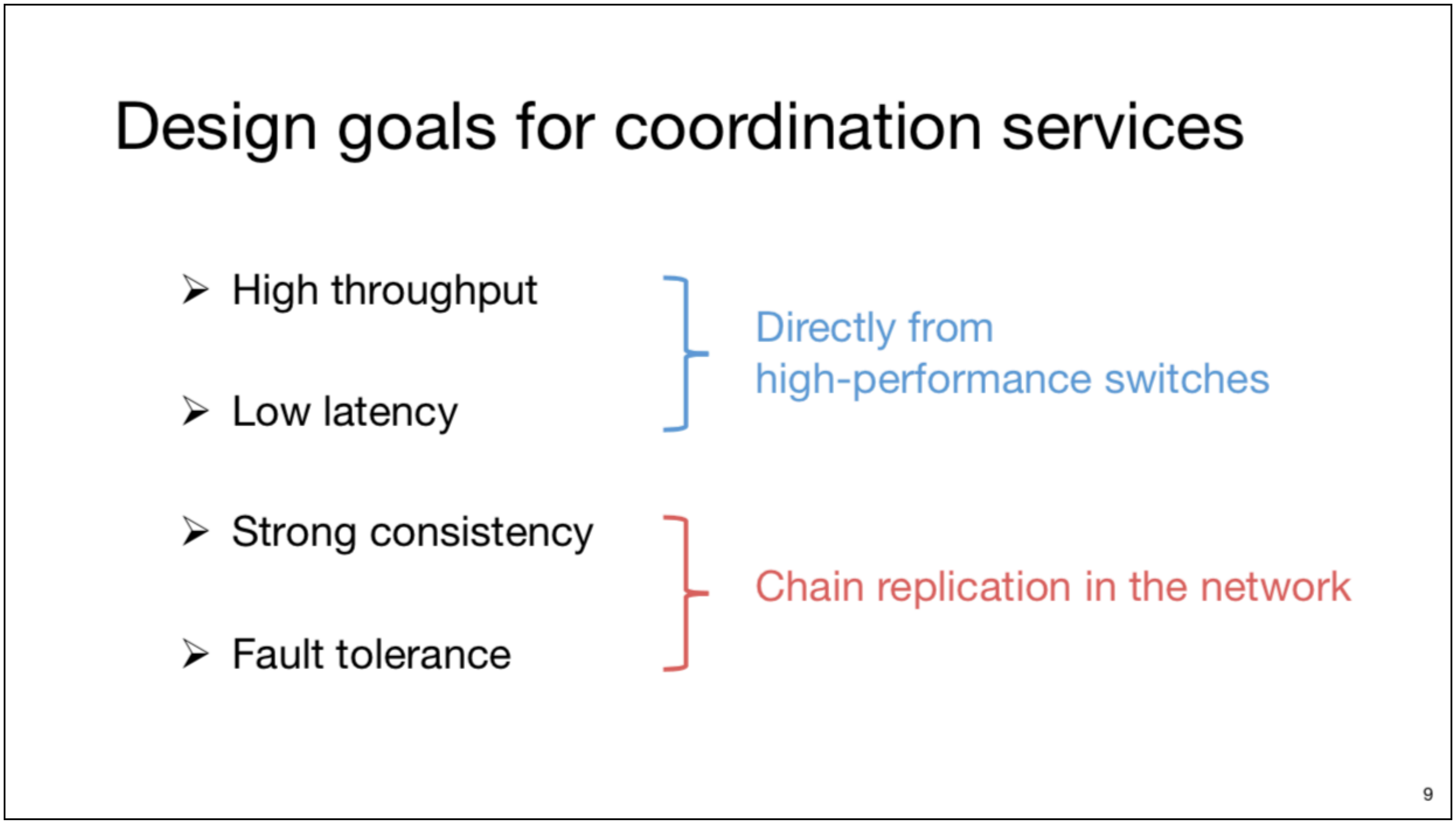
Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

State of the art server-based coordination services struggle to provide high-throughput and low-latency



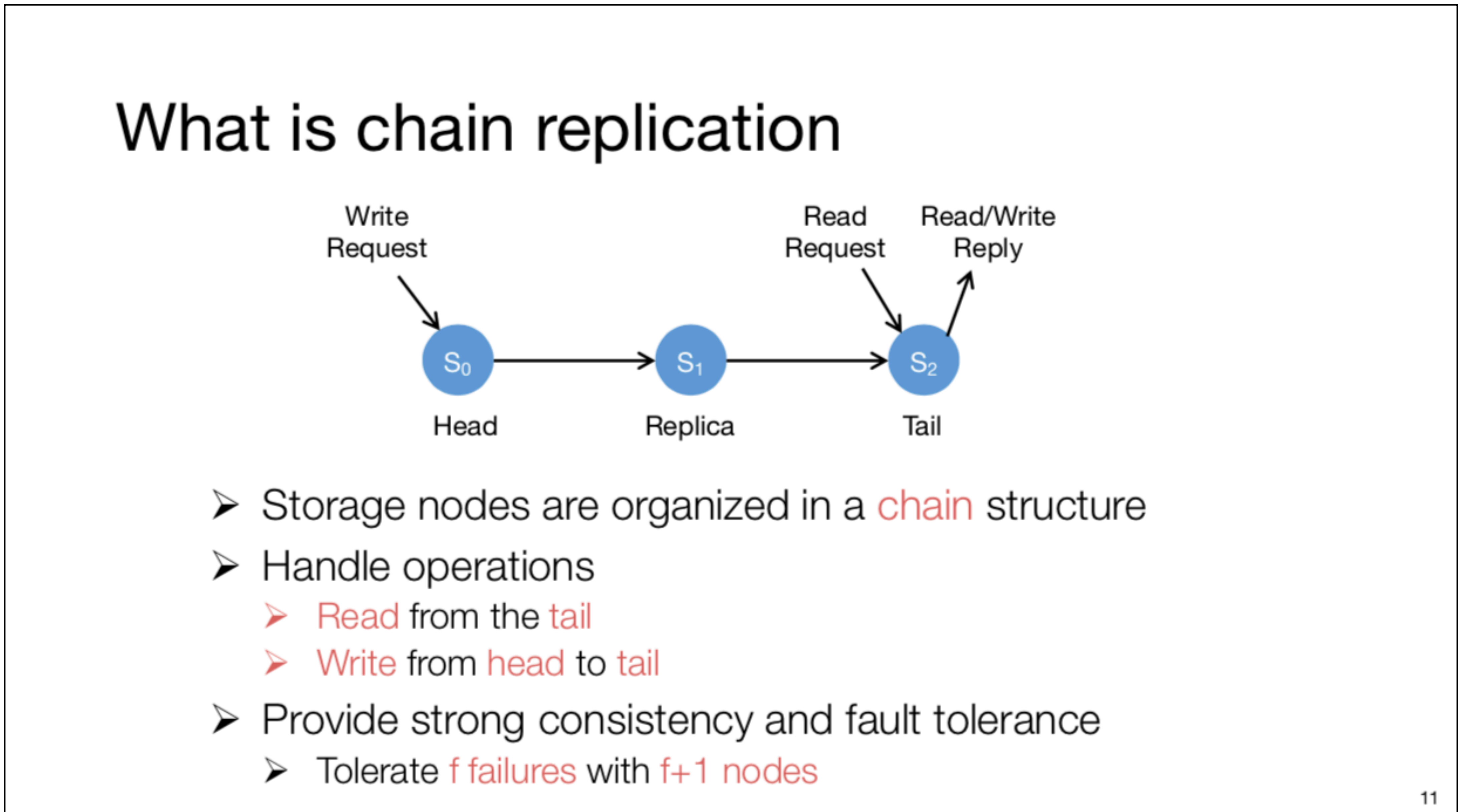
Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

Key challenge is to ensure consistency and fault-tolerance



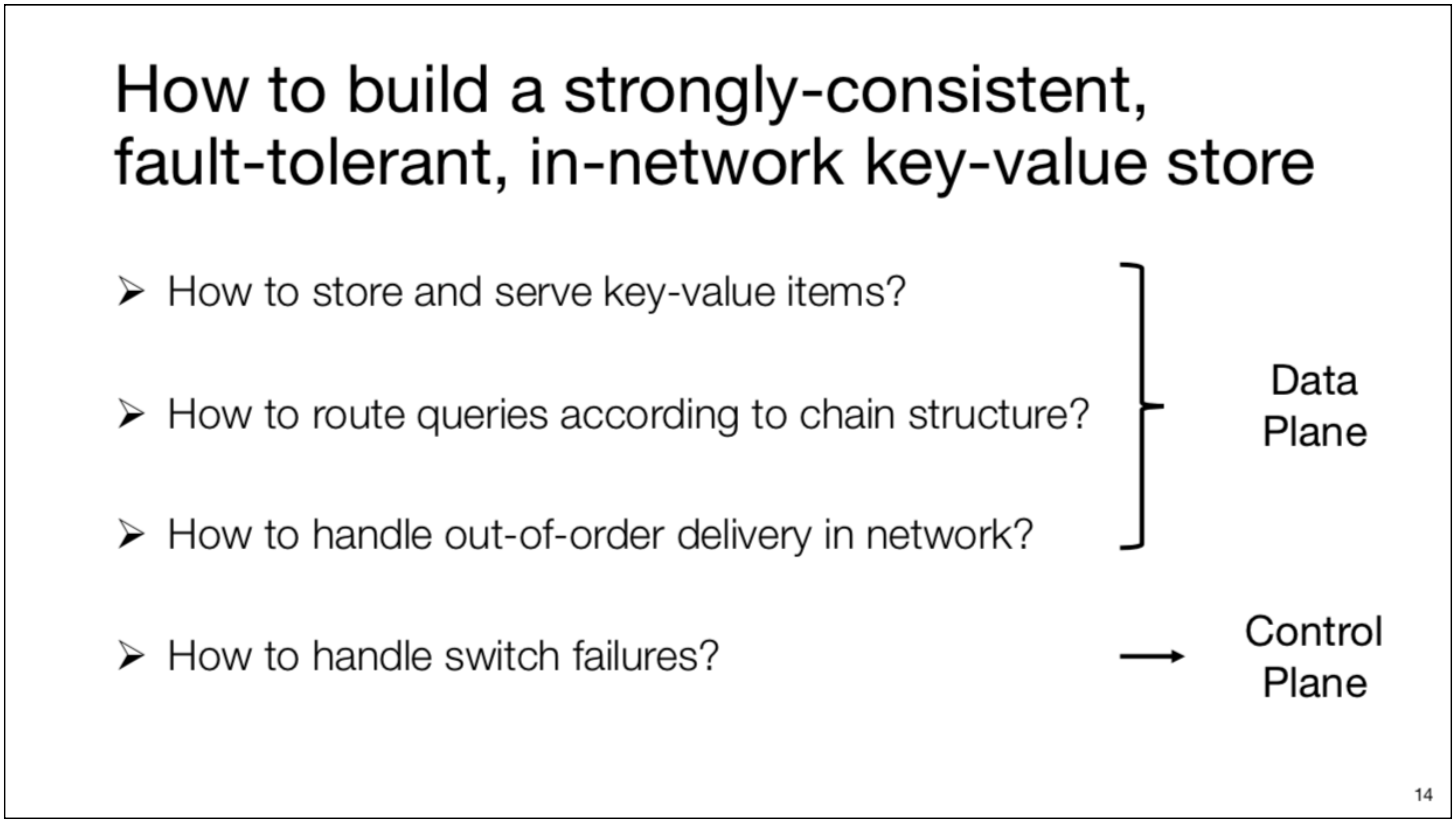
Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

NetChain does so using chain replication, building upon NetCache for storing values in each switch



Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

NetChain relies on a tailored UDP-based protocol, source-routing mechanisms and message serialization



Source: NetChain: Scale-Free Sub-RTT Coordination, Xin Jin, 2018

This week on

Advanced Topics in Communication Networks

A high-level, non-exhaustive overview of the research surrounding data plane programmability

A high-level, non-exhaustive overview of the research surrounding data plane programmability

Data plane programmability for

- Performance
- Monitoring
- Applications offloading

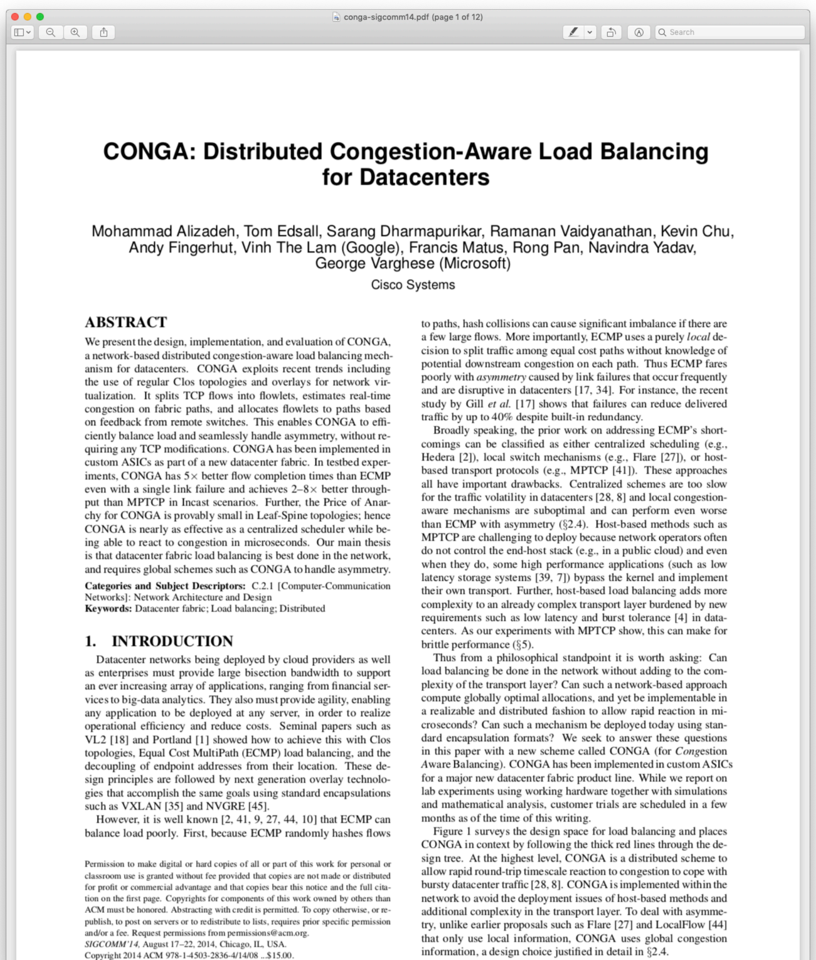
- Platforms
- Correctness
- Management

for Data plane programmability

Data plane programmability	for	Performance Monitoring Applications offloading

Platforms Correctness Management	for	Data plane programmability

A large set of papers on programmable data planes aim at improving performance, esp. load balancing



CONGA [SIGCOMM'14]

HULA [SOSR'16]

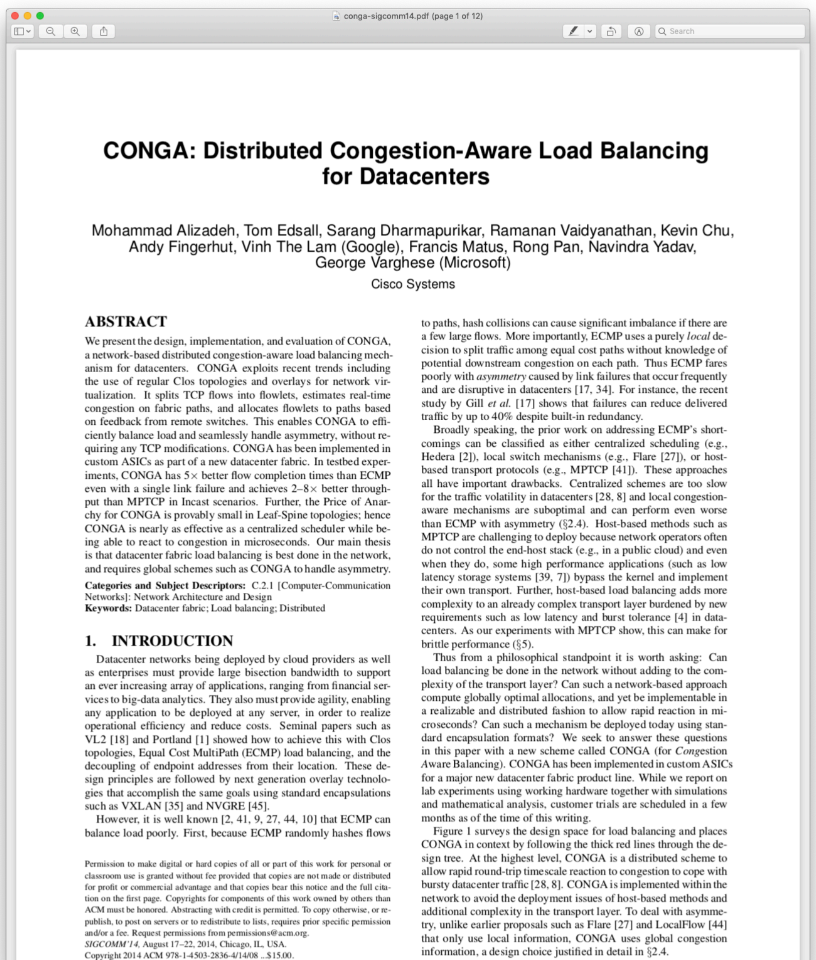
DRILL [SIGCOMM'17]

LetFlow [NSDI'17]

A large set of papers on programmable data planes aim at improving performance, esp. load balancing

HULA [SOSR'16]

DRILL [SIGCOMM'17]



conga-sigcomm14.pdf (page 1 of 12)

CONGA: Distributed Congestion-Aware Load Balancing for Datacenters

Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu,
Andy Fingerhut, Vinh The Lam (Google), Francis Matus, Rong Pan, Navindra Yadav,
George Varghese (Microsoft)
Cisco Systems

ABSTRACT

We present the design, implementation, and evaluation of CONGA, a network-based distributed congestion-aware load balancing mechanism for datacenters. CONGA exploits recent trends including the use of regular Clos topologies and overlays for network virtualization. It splits TCP flows into flowlets, estimates real-time congestion on fabric paths, and allocates flowlets to paths based on feedback from remote switches. This enables CONGA to efficiently balance load and seamlessly handle asymmetry, without requiring any TCP modifications. CONGA has been implemented in custom ASICs as part of a new datacenter fabric. In testbed experiments, CONGA has 5× better flow completion times than ECMP even with a single link failure and achieves 2–8× better throughput than MPTCP in Incast scenarios. Further, the Price of Anarchy for CONGA is provably small in Leaf-Spine topologies; hence CONGA is nearly as effective as a centralized scheduler while being able to react to congestion in microseconds. Our main thesis is that datacenter fabric load balancing is best done in the network, and requires global schemes such as CONGA to handle asymmetry.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design
Keywords: Datacenter fabric; Load balancing; Distributed

1. INTRODUCTION

Datacenter networks being deployed by cloud providers as well as enterprises must provide large bisection bandwidth to support an ever increasing array of applications, ranging from financial services to big-data analytics. They also must provide agility, enabling any application to be deployed at any server, in order to realize operational efficiency and reduce costs. Seminal papers such as VL2 [18] and Portland [1] showed how to achieve this with Clos topologies, Equal Cost MultiPath (ECMP) load balancing, and the decoupling of endpoint addresses from their location. These design principles are followed by next generation overlay technologies that accomplish the same goals using standard encapsulations such as VXLAN [35] and NVGRE [45].

However, it is well known [2, 41, 9, 27, 44, 10] that ECMP can balance load poorly. First, because ECMP randomly hashes flows to paths, hash collisions can cause significant imbalance if there are a few large flows. More importantly, ECMP uses a purely *local* decision to split traffic among equal cost paths without knowledge of potential downstream congestion on each path. Thus ECMP fares poorly with *asymmetry* caused by link failures that occur frequently and are disruptive in datacenters [17, 34]. For instance, the recent study by Gill *et al.* [17] shows that failures can reduce delivered traffic by up to 40% despite built-in redundancy.

Broadly speaking, the prior work on addressing ECMP's shortcomings can be classified as either centralized scheduling (e.g., Hedera [2]), local switch mechanisms (e.g., Flare [27]), or host-based transport protocols (e.g., MPTCP [41]). These approaches all have important drawbacks. Centralized schemes are too slow for the traffic volatility in datacenters [28, 8] and local congestion-aware mechanisms are suboptimal and can perform even worse than ECMP with asymmetry (§2.4). Host-based methods such as MPTCP are challenging to deploy because network operators often do not control the end-host stack (e.g., in a public cloud) and even when they do, some high performance applications (such as low latency storage systems [39, 7]) bypass the kernel and implement their own transport. Further, host-based load balancing adds more complexity to an already complex transport layer burdened by new requirements such as low latency and burst tolerance [4] in datacenters. As our experiments with MPTCP show, this can make for brittle performance (§5).

Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow rapid reaction in microseconds? Can such a mechanism be deployed today using standard encapsulation formats? We seek to answer these questions in this paper with a new scheme called CONGA (for *Cong*estion *A*ware Balancing). CONGA has been implemented in custom ASICs for a major new datacenter fabric product line. While we report on lab experiments using working hardware together with simulations and mathematical analysis, customer trials are scheduled in a few months as of the time of this writing.

Figure 1 surveys the design space for load balancing and places CONGA in context by following the thick red lines through the design tree. At the highest level, CONGA is a distributed scheme to allow rapid round-trip timescale reaction to congestion to cope with bursty datacenter traffic [28, 8]. CONGA is implemented within the network to avoid the deployment issues of host-based methods and additional complexity in the transport layer. To deal with asymmetry, unlike earlier proposals such as Flare [27] and LocalFlow [44] that only use local information, CONGA uses global congestion information, a design choice justified in detail in §2.4.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
Copyright 2014 ACM 978-1-4503-2836-4/14/08 ...$15.00.
http://dx.doi.org/10.1145/2619239.2626316 .

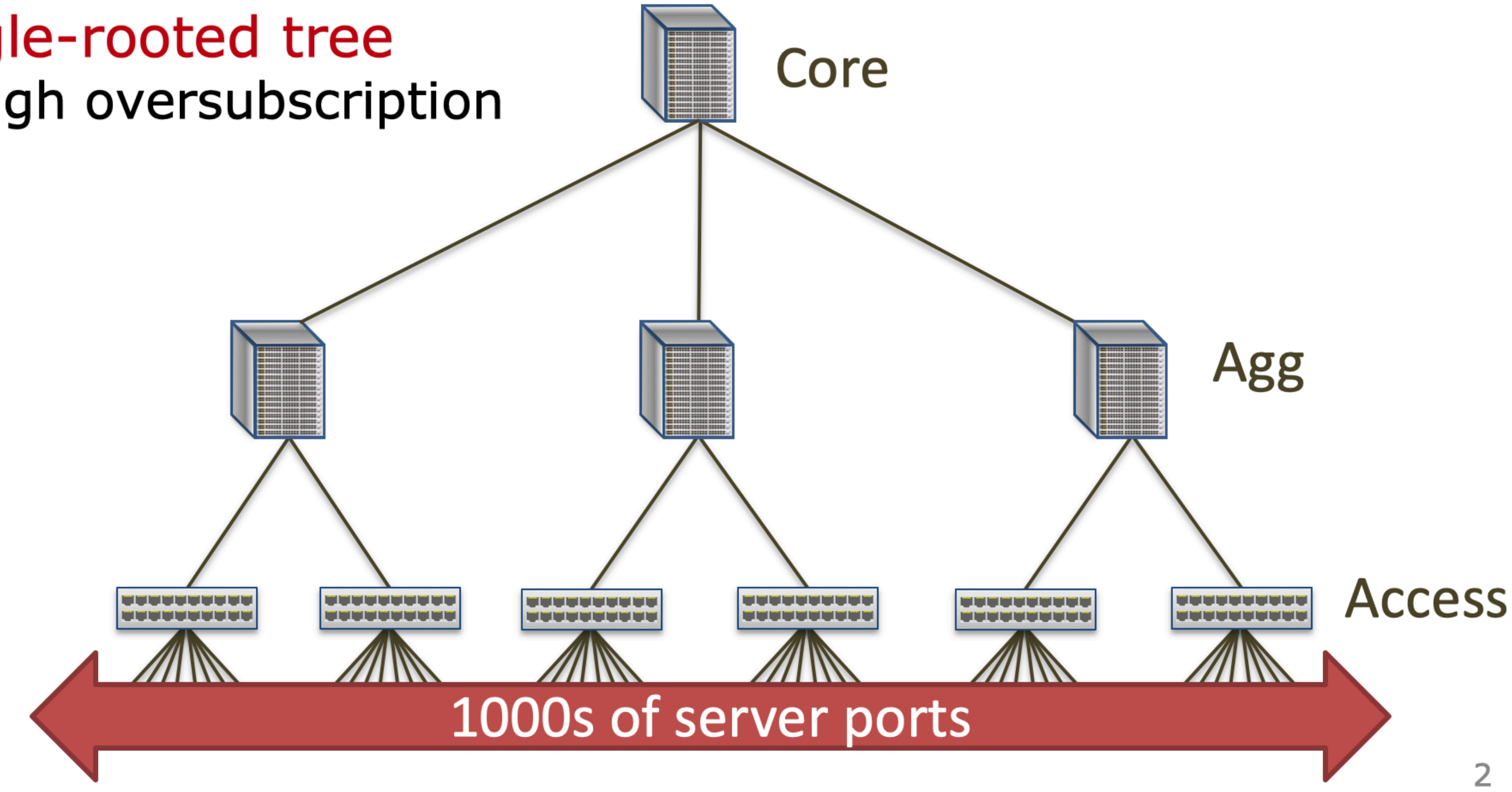
CONGA [SIGCOMM'14]

LetFlow [NSDI'17]

Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

Single-rooted tree
- ➢ High oversubscription



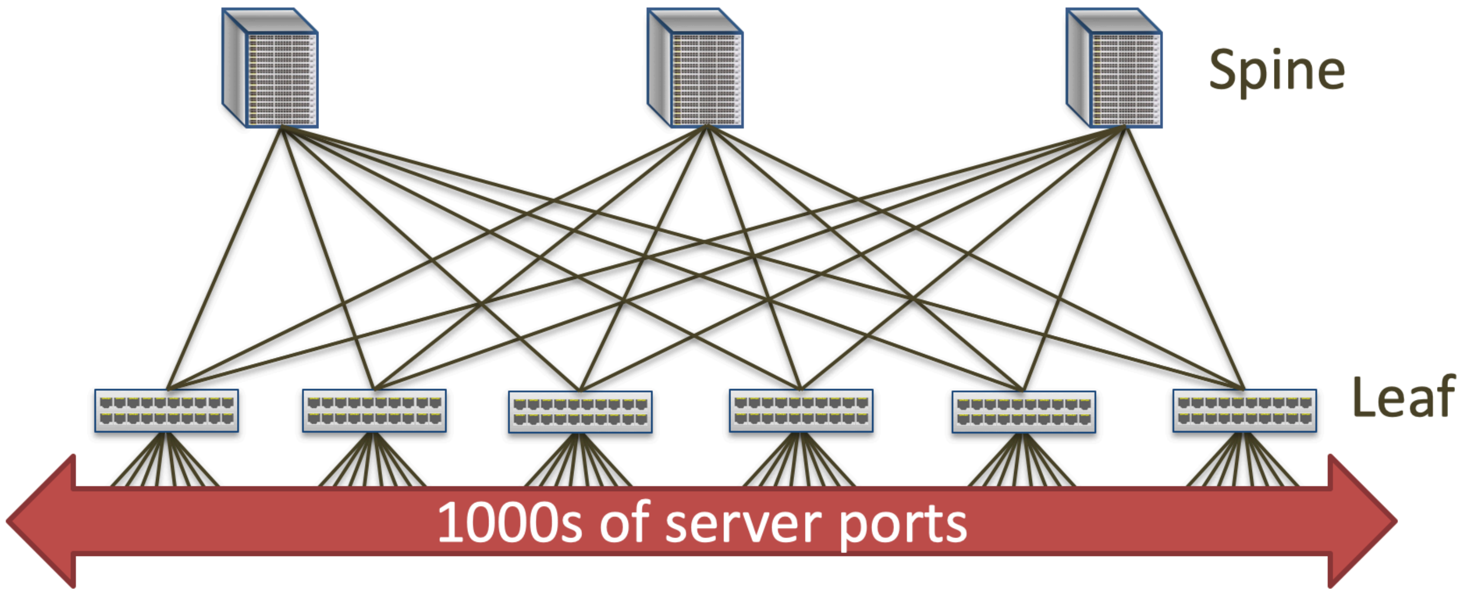
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

Multi-rooted tree [Fat-tree, Leaf-Spine, …]

- Full bisection bandwidth, achieved via multipathing

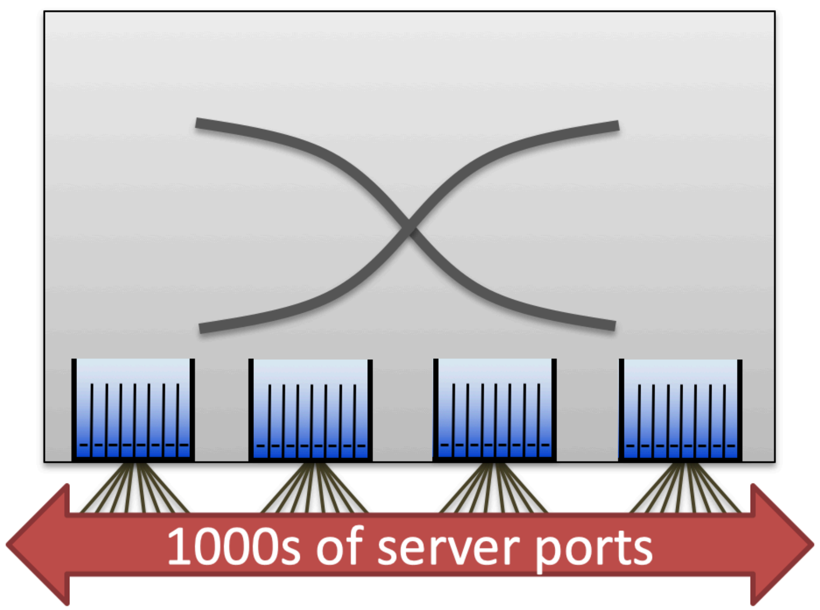


Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

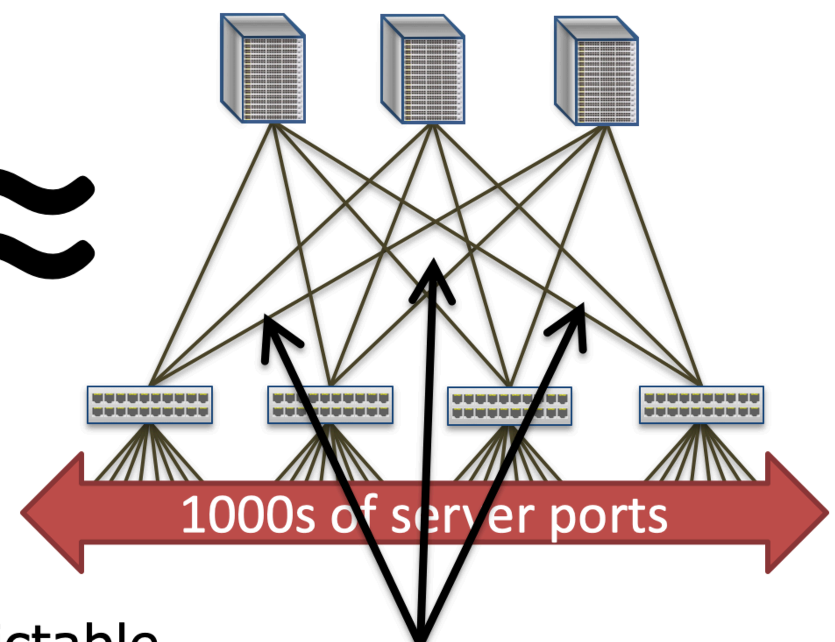
Multi-rooted != Ideal DC Network

Ideal DC network:
Big output-queued switch

Multi-rooted tree



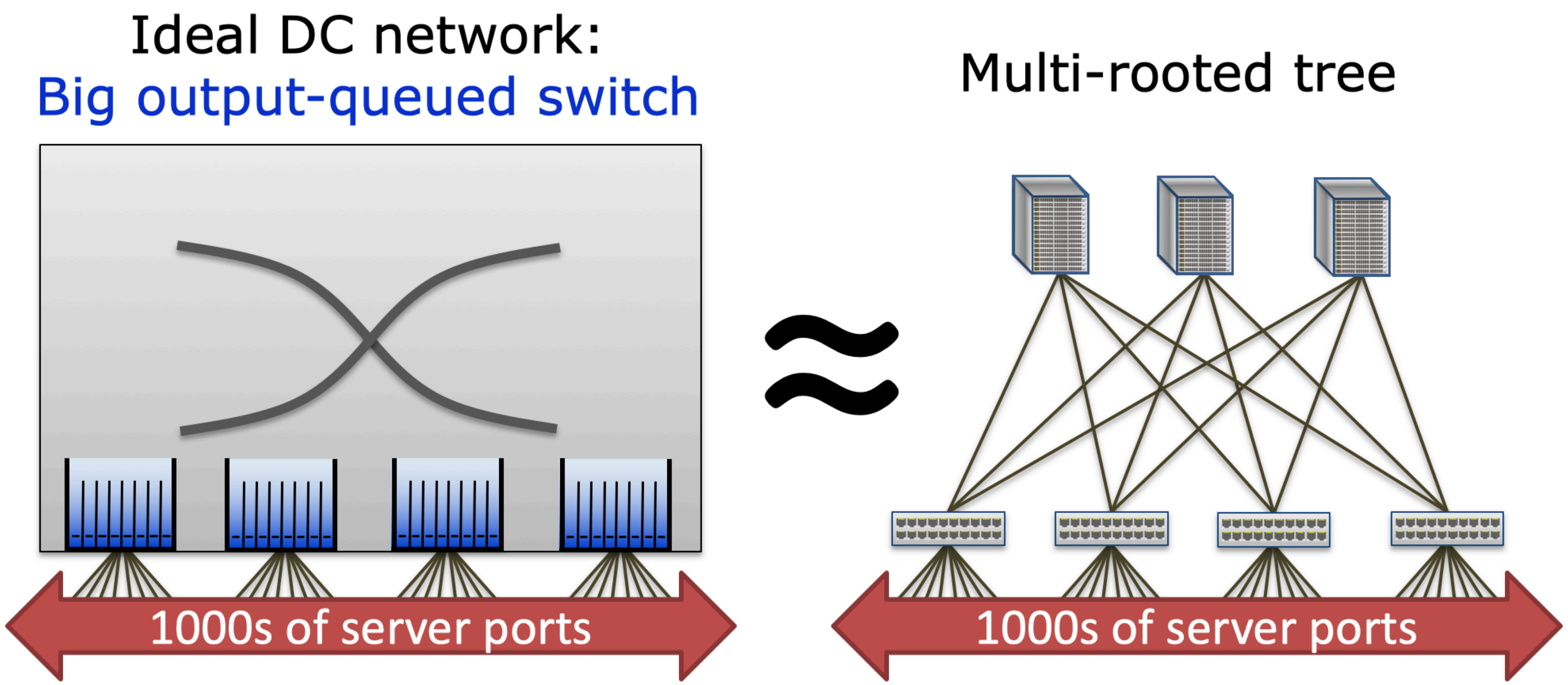
≈



- No internal bottlenecks ➔ predictable
- Simplifies BW management
  [EyeQ, FairCloud, pFabric, Varys, …]

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Multi-rooted != Ideal DC Network



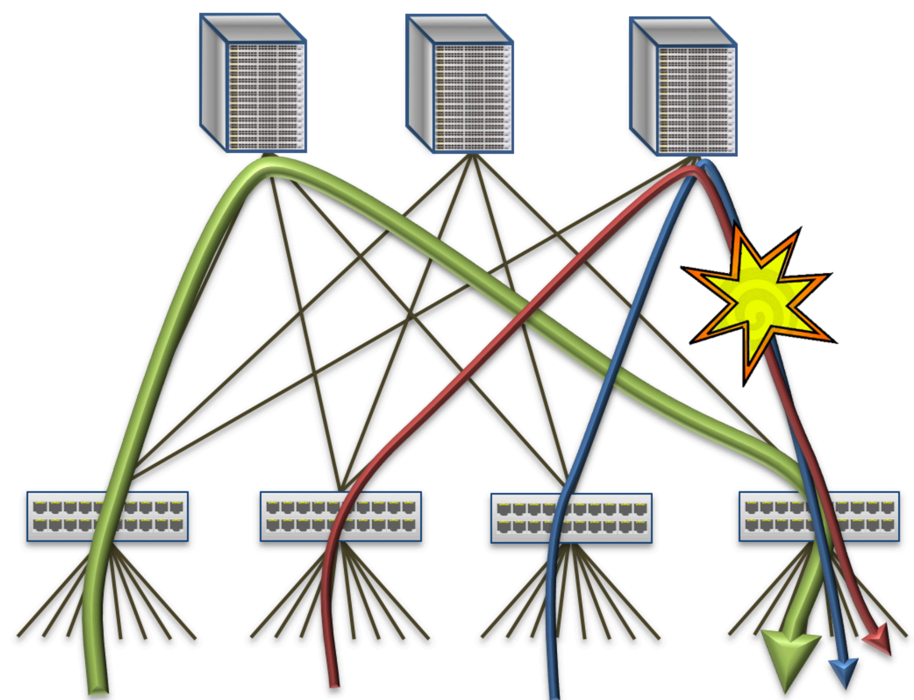
Need precise load balancing

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Today: ECMP Load Balancing

Pick among equal-cost paths by a hash of 5-tuple

- ➢ Approximates Valiant load balancing
- ➢ Preserves packet order



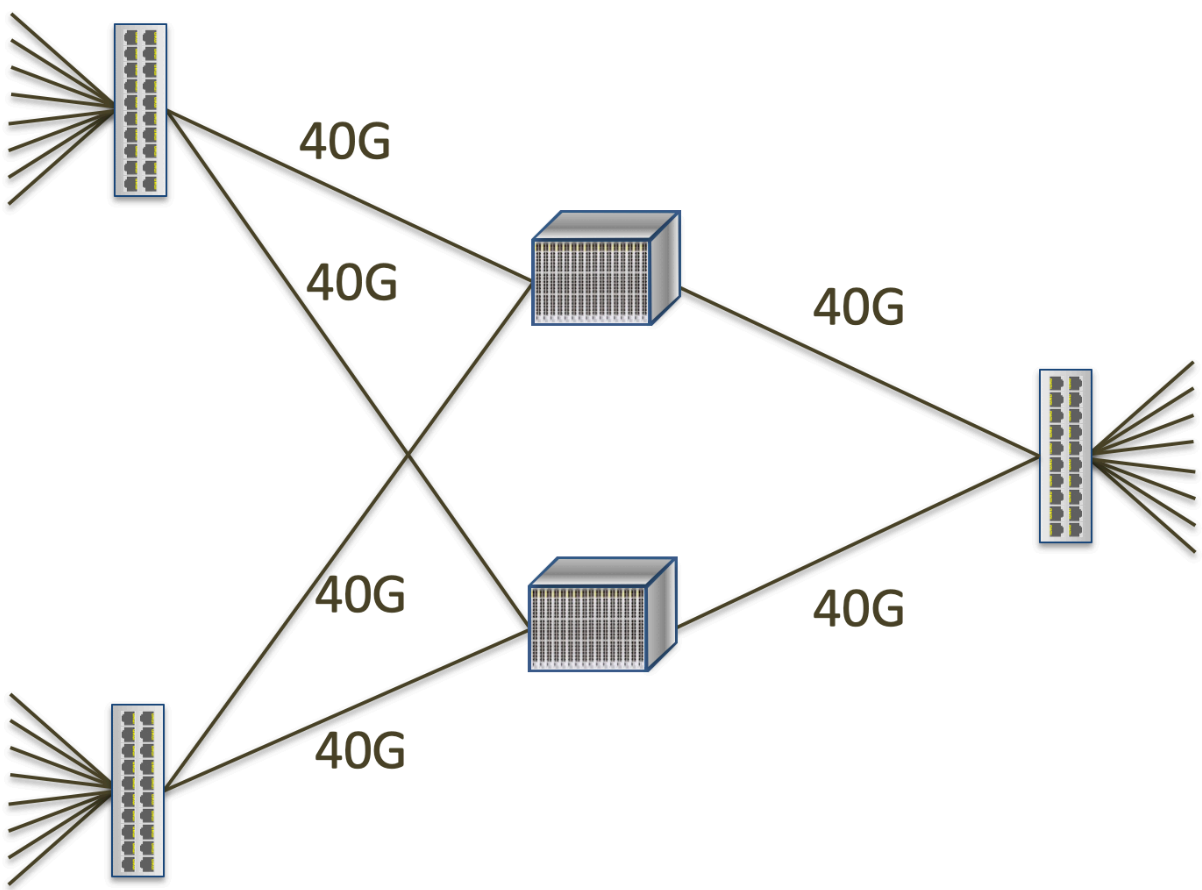
Problems:

- Hash collisions
  (coarse granularity)
- Local & stateless
  (v. bad with asymmetry
  due to link failures)

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Dealing with Asymmetry

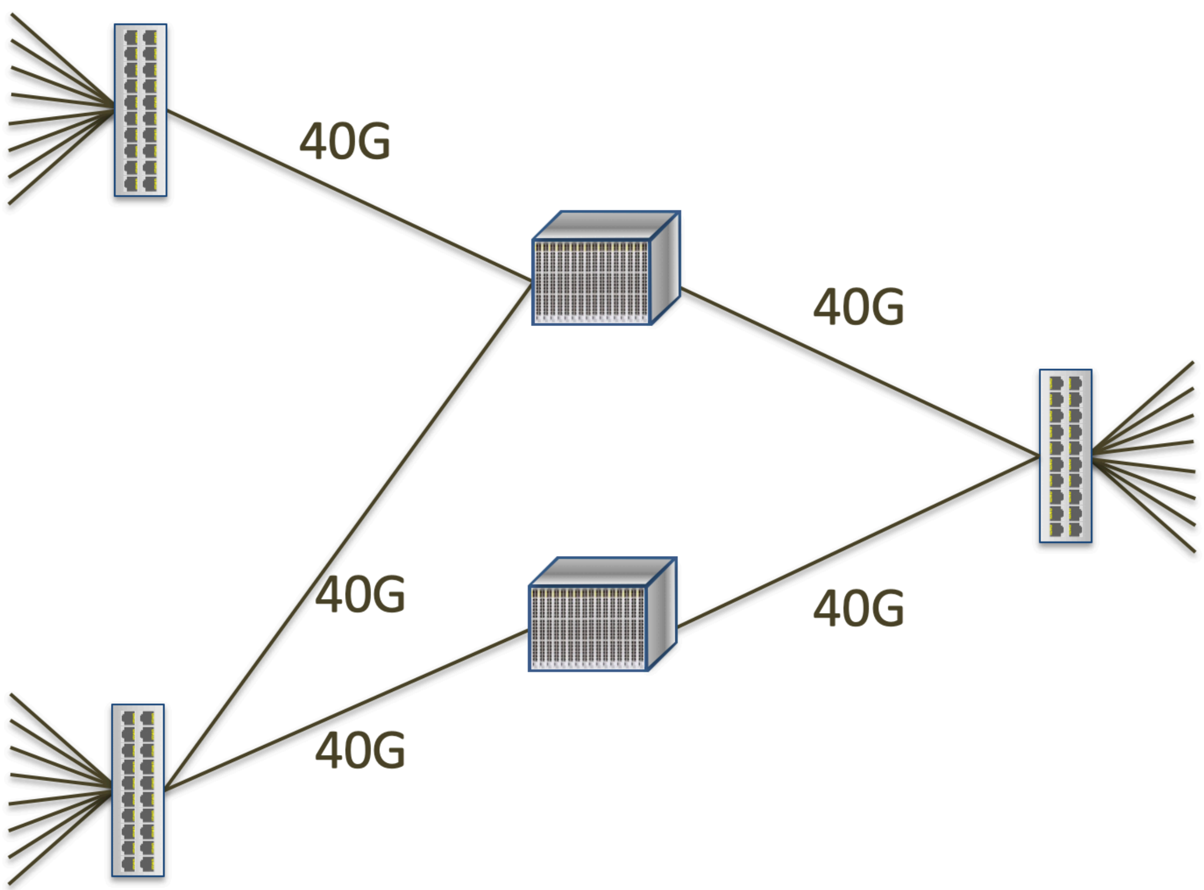
Handling asymmetry needs non-local knowledge



Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

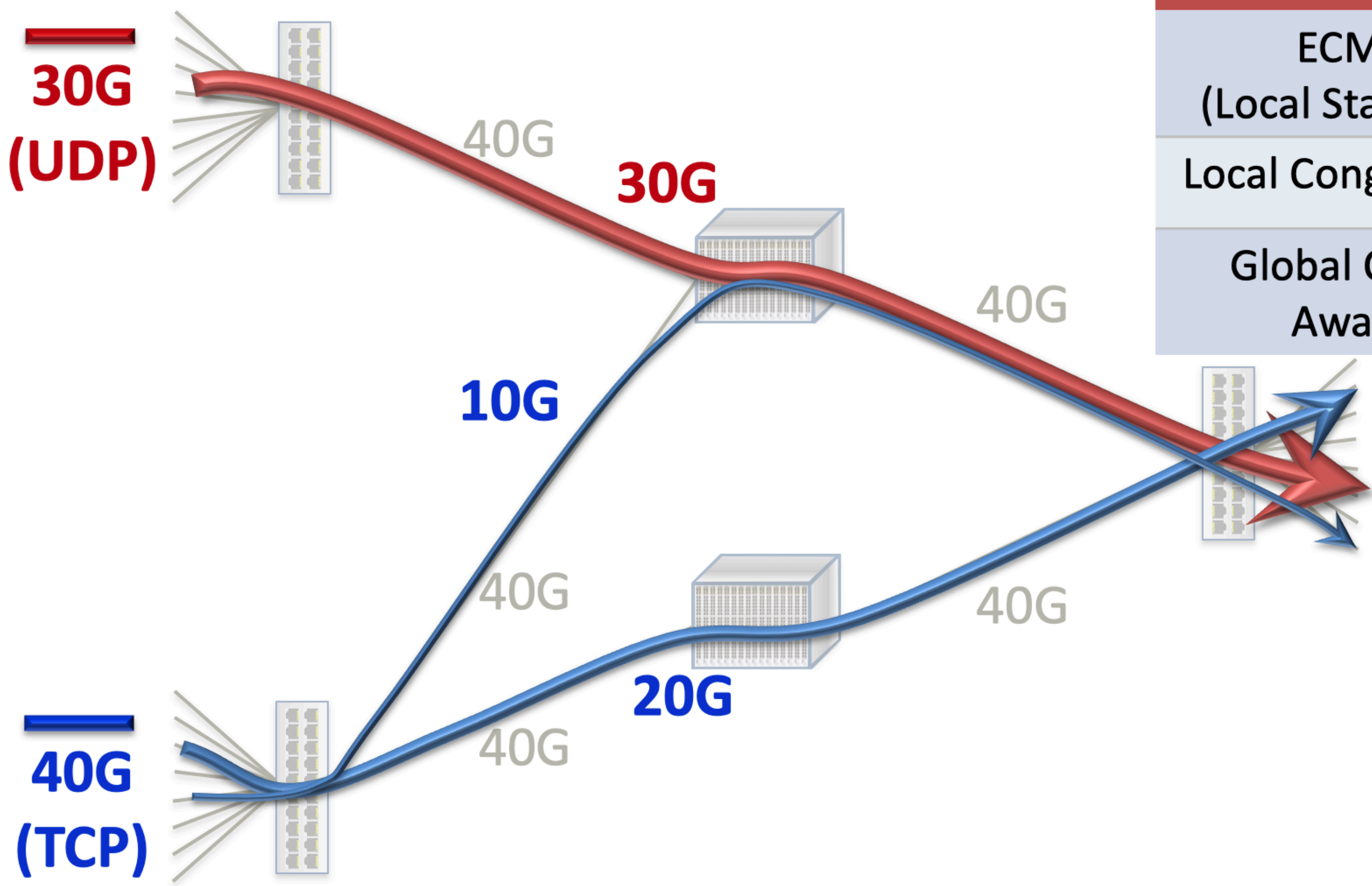
Dealing with Asymmetry

Handling asymmetry needs non-local knowledge



Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

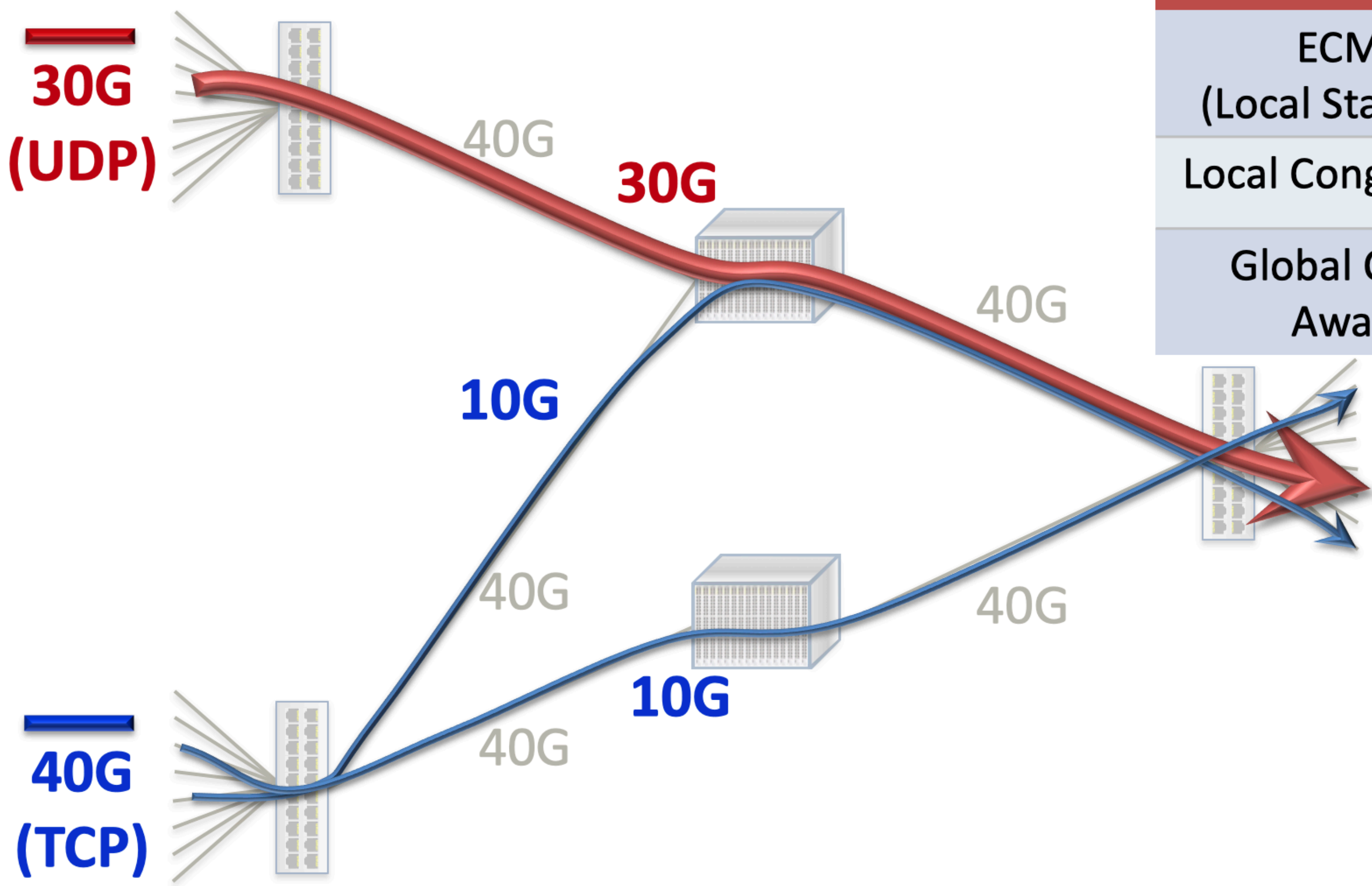
Dealing with Asymmetry: ECMP



Scheme	Thrput
ECMP (Local Stateless)	**60G**
Local Cong-Aware	
Global Cong-Aware	

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

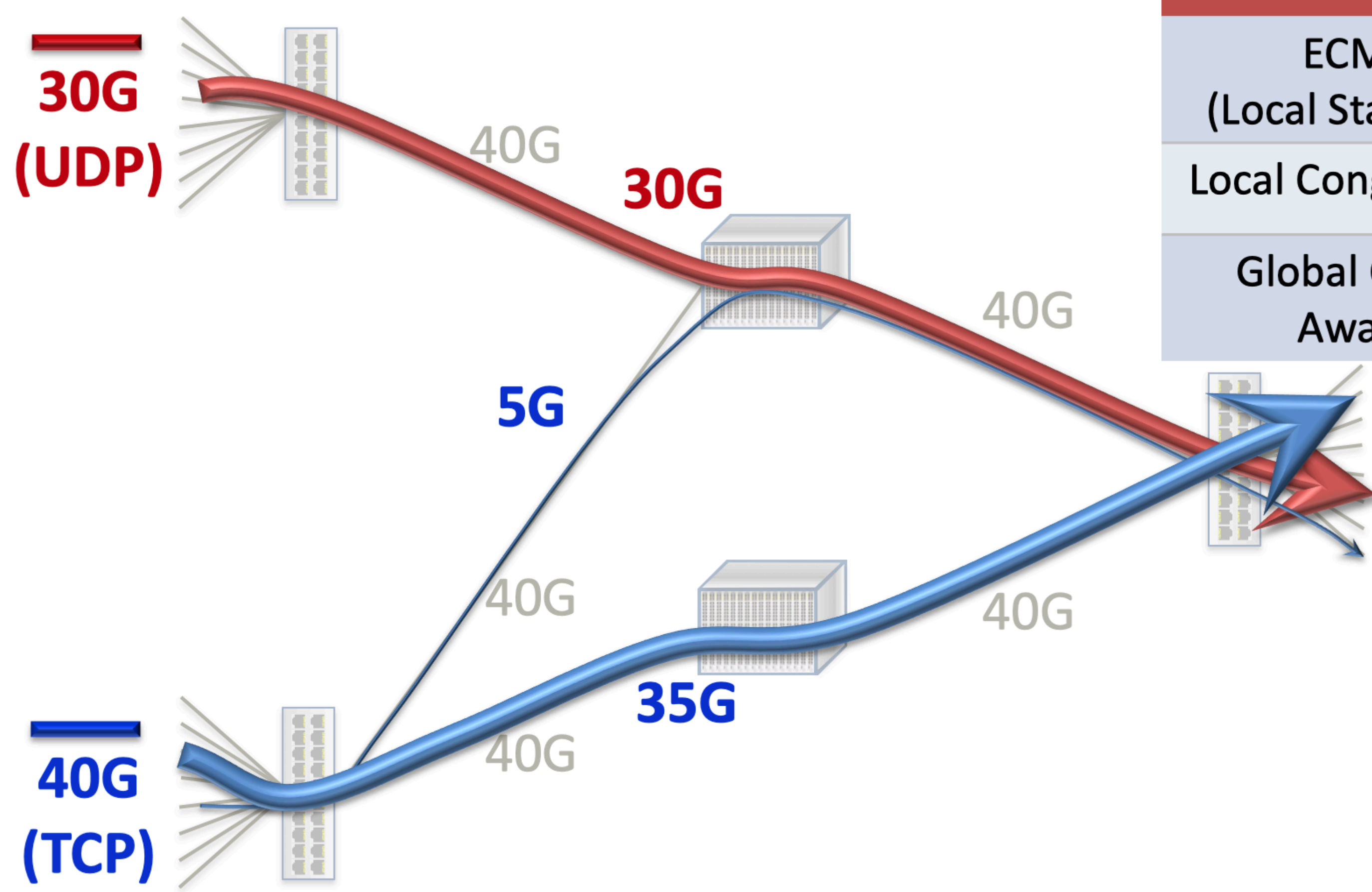
Dealing with Asymmetry: Local Congestion-Aware



Scheme	Thrput
ECMP (Local Stateless)	**60G**
Local Cong-Aware	**50G**
Global Cong-Aware	

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

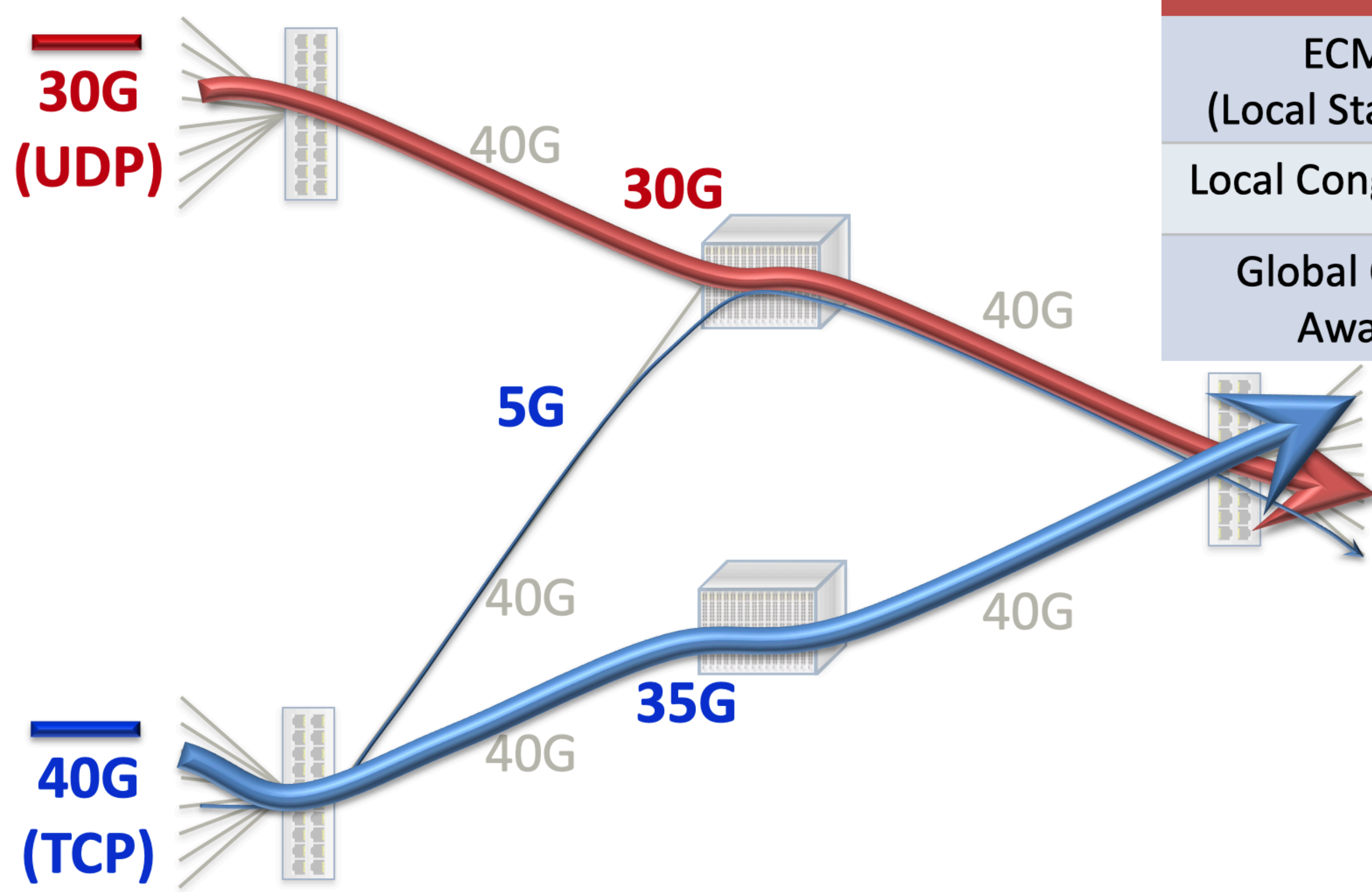
Dealing with Asymmetry: Global Congestion-Aware



Scheme	Thrput
ECMP (Local Stateless)	60G
Local Cong-Aware	50G
Global Cong-Aware	70G

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Dealing with Asymmetry: Global Congestion-Aware



Scheme	Thrput
ECMP (Local Stateless)	**60G**
Local Cong-Aware	**50G**
Global Cong-Aware	**70G**

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Global Congestion-Awareness (in Datacenters)

		Datacenter
Opportunity →	Latency	microseconds
	Topology	simple, regular
Challenge →	Traffic	volatile, bursty

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Global Congestion-Awareness (in Datacenters)

		Datacenter
Opportunity →	Latency	microseconds
	Topology	simple, regular
Challenge →	Traffic	volatile, bursty

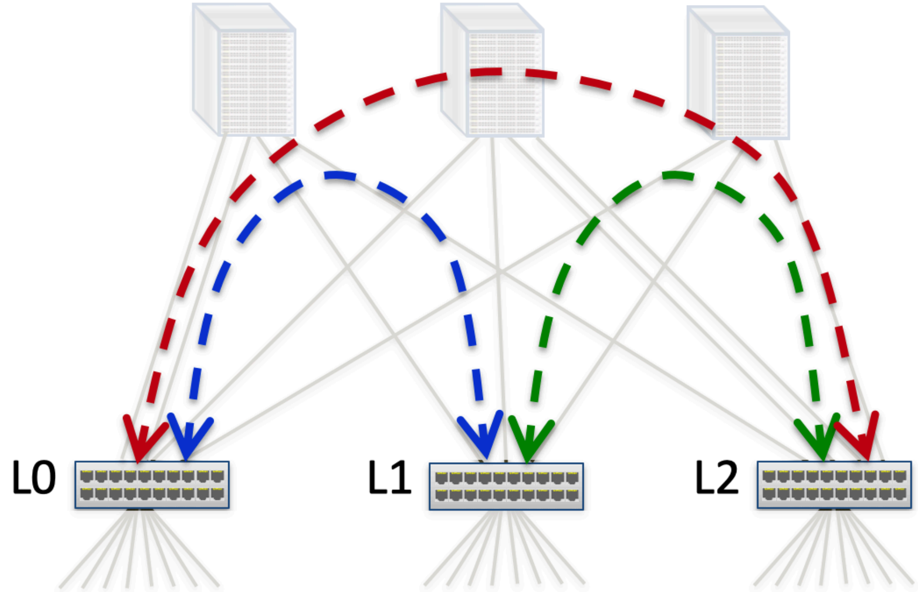
Key Insight:
Use *extremely* fast, low latency distributed control

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

CONGA in 1 Slide

1. Leaf switches (top-of-rack) track congestion to other leaves on different paths in near real-time

1. Use greedy decisions to minimize bottleneck util



Fast feedback loops between leaf switches, directly in dataplane

Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

A large set of papers on programmable data planes aim at improving performance, esp. load balancing

P4-based data-plane load-balancing with better scalability than CONGA — HULA [SOSR'16]

"micro" load balancing, packet-by-packet, can deal with micro-bursts — DRILL [SIGCOMM'17]

stateless, yet congestion-aware load-balancing decision — LetFlow [NSDI'17]

Data plane programmability | for | Performance
Monitoring
Applications offloading

Platforms
Correctness
Management | for | Data plane programmability

Language-Directed Hardware Design for Network Performance Monitoring

Sonata: Query-Driven Streaming Network Telemetry

Arpit Gupta, Princeton University; Rob Harrison, Princeton University; Marco Canini, KAUST; Nick Feamster, Princeton University; Jennifer Rexford, Princeton University; Walter Willinger, NIKSUN Inc.

LossRadar: Fast Detection of Lost Packets in Data Center Networks

FlowRadar: A Better NetFlow for Data Centers

Yuliang Li*, Rui Miao*, Changhoon Kim†, Minlan Yu*
*University of Southern California †Barefoot Networks

Dapper: Data Plane Performance Diagnosis of TCP

Network-Wide Heavy Hitter Detection with Commodity Switches

Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford
Princeton University

In-band Network Telemetry (INT)

June 2016

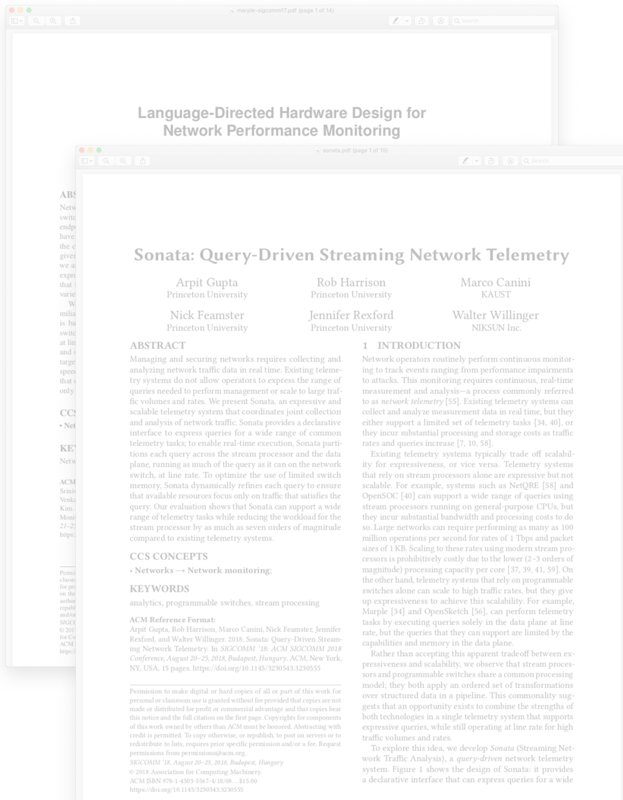
Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

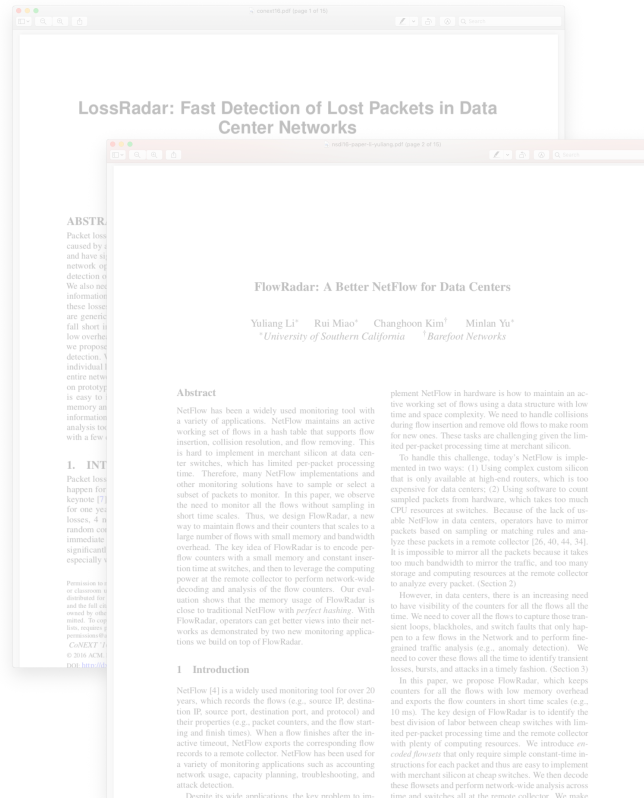
Elastic Sketch: Adaptive and Fast Network-wide Measurements

One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon

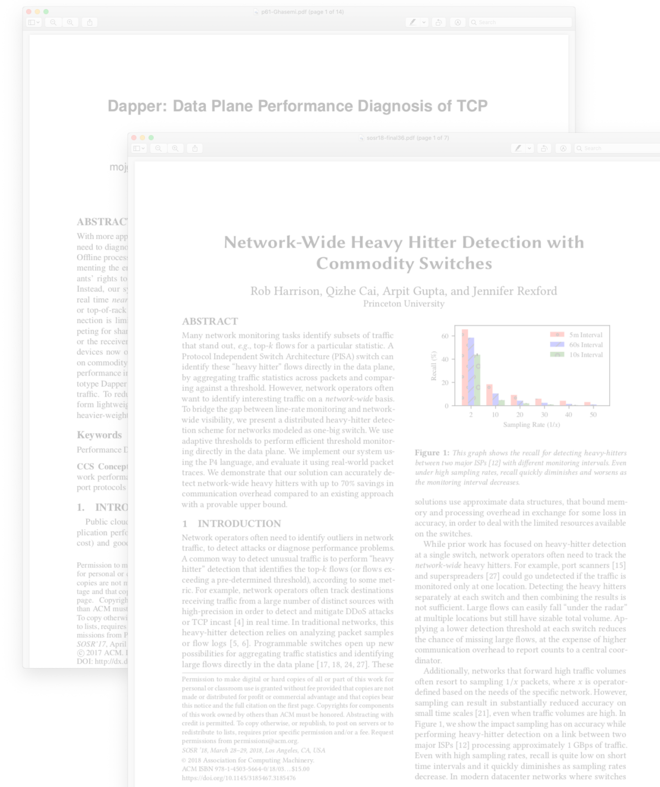
Zaoxing Liu†, Antonis Manousis*, Gregory Vorsanger†, Vyas Sekar*, Vladimir Braverman†
† Johns Hopkins University * Carnegie Mellon University


Language-Directed Hardware Design for Network Performance Monitoring

Sonata: Query-Driven Streaming Network Telemetry

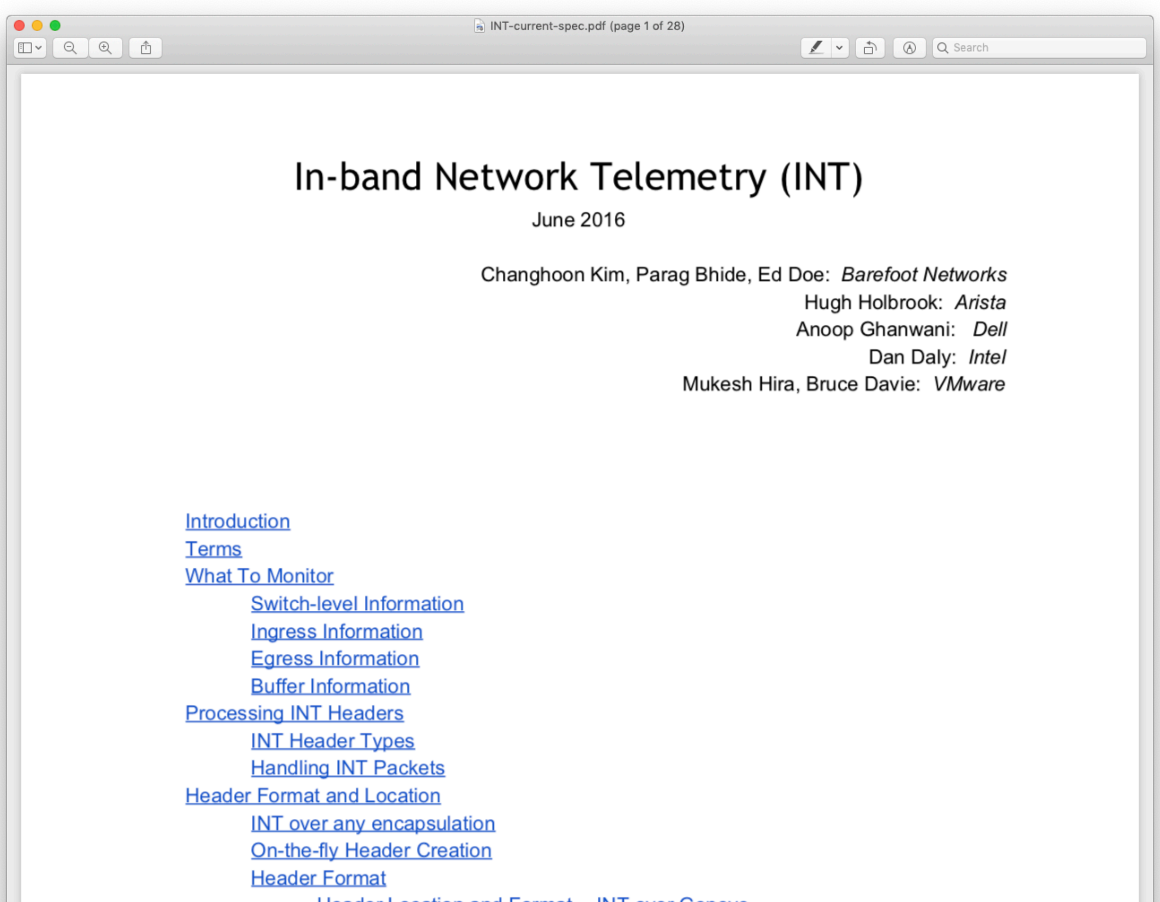

LossRadar: Fast Detection of Lost Packets in Data Center Networks

FlowRadar: A Better NetFlow for Data Centers


Dapper: Data Plane Performance Diagnosis of TCP

Network-Wide Heavy Hitter Detection with Commodity Switches

Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford


INT-current-spec.pdf (page 1 of 28)

In-band Network Telemetry (INT)

June 2016

Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

Introduction
Terms
What To Monitor
- Switch-level Information
- Ingress Information
- Egress Information
- Buffer Information

Processing INT Headers
- INT Header Types
- Handling INT Packets

Header Format and Location
- INT over any encapsulation
- On-the-fly Header Creation
- Header Format
  - Header Location and Format -- INT over Geneve


SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Elastic Sketch: Adaptive and Fast Network-wide Measurements

One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon

Current monitoring methods are inadequate

- Not fast enough
  - Involve CPU and control planes
  - Network state changes rapidly

- Do not provide end-to-end state
  - Difficult to correlate per-element state with the actual path of a flow

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

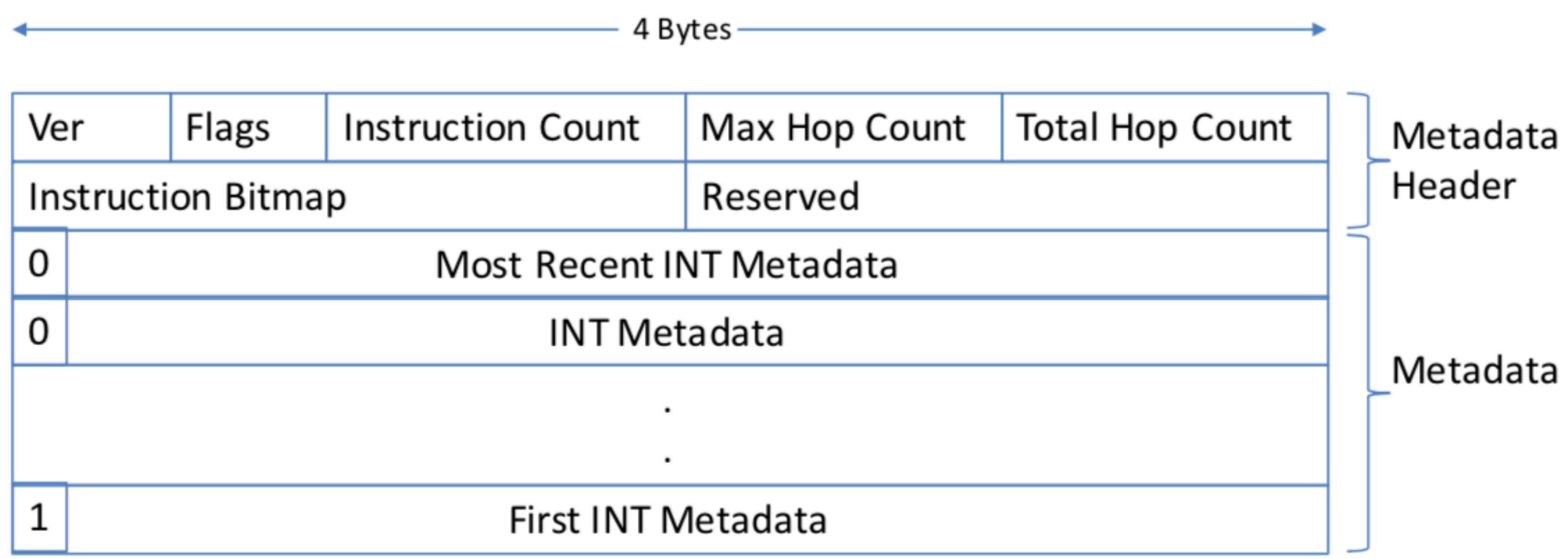
INT : In-band Network Telemetry

- Mechanism for collecting network state in the dataplane
  - As close to realtime as possible
  - At current and future line rates
  - With a framework that can adapt over time

- Examples of network state
  - Switch ID, Ingress Port ID, Egress Port ID
  - Egress Link Utilization
  - Hop Latency
  - Egress Queue Occupancy
  - Egress Queue Congestion Status
  - ....

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

INT Header Format



Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

INT using P4

- P4 enables flexible packet parsing and modification for INT

- P4 allows INT to adapt to
  - Any Encapsulation format
  - Any State required to be collected
  - Any feature, protocol – current and future

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

INT : P4 Code Snippet

Exact-match
Table Definition

```
table int_inst {
  reads {
    int_header.instruction_mask : exact;
  }
  actions {
    int_set_header_i0;
    int_set_header_i1;
    int_set_header_i2;
    int_set_header_i3;
    .....
  }
```

Action
Definitions

```
action int_set_header_i0() {
}
action int_set_header_i1() {
  int_set_header_3();
}
action int_set_header_i2() {
  int_set_header_2();
}
action int_set_header_i3() {
  int_set_header_3();
  int_set_header_2();
}
.....
```

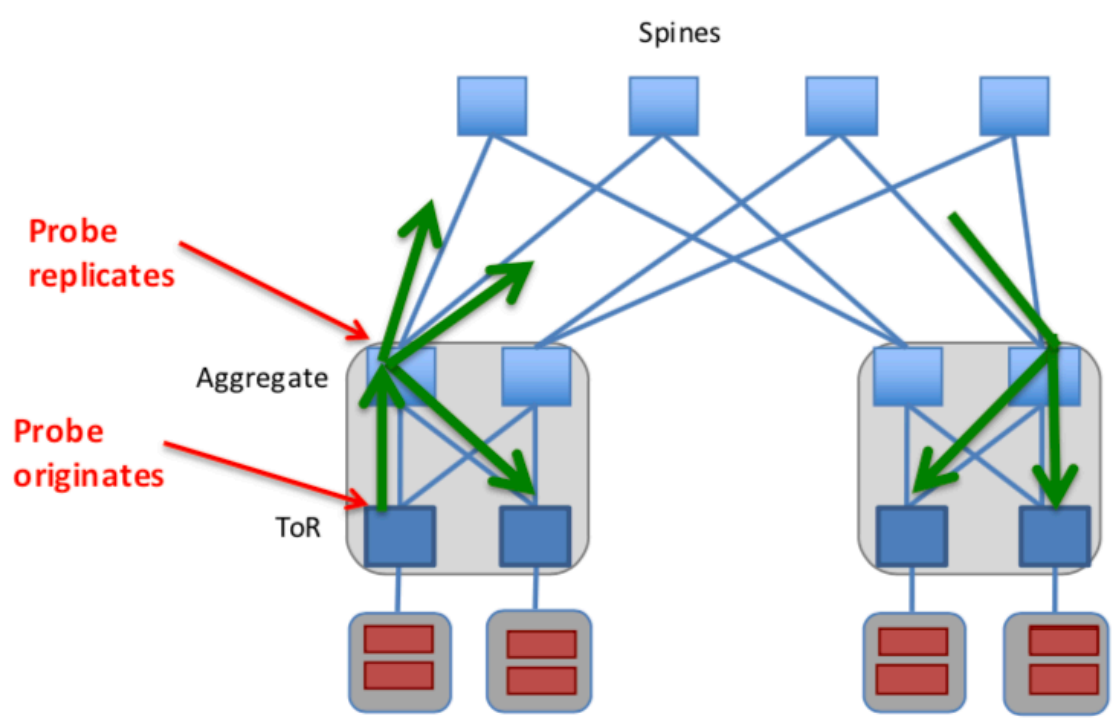
Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

HULA: INT + Flowlet routing

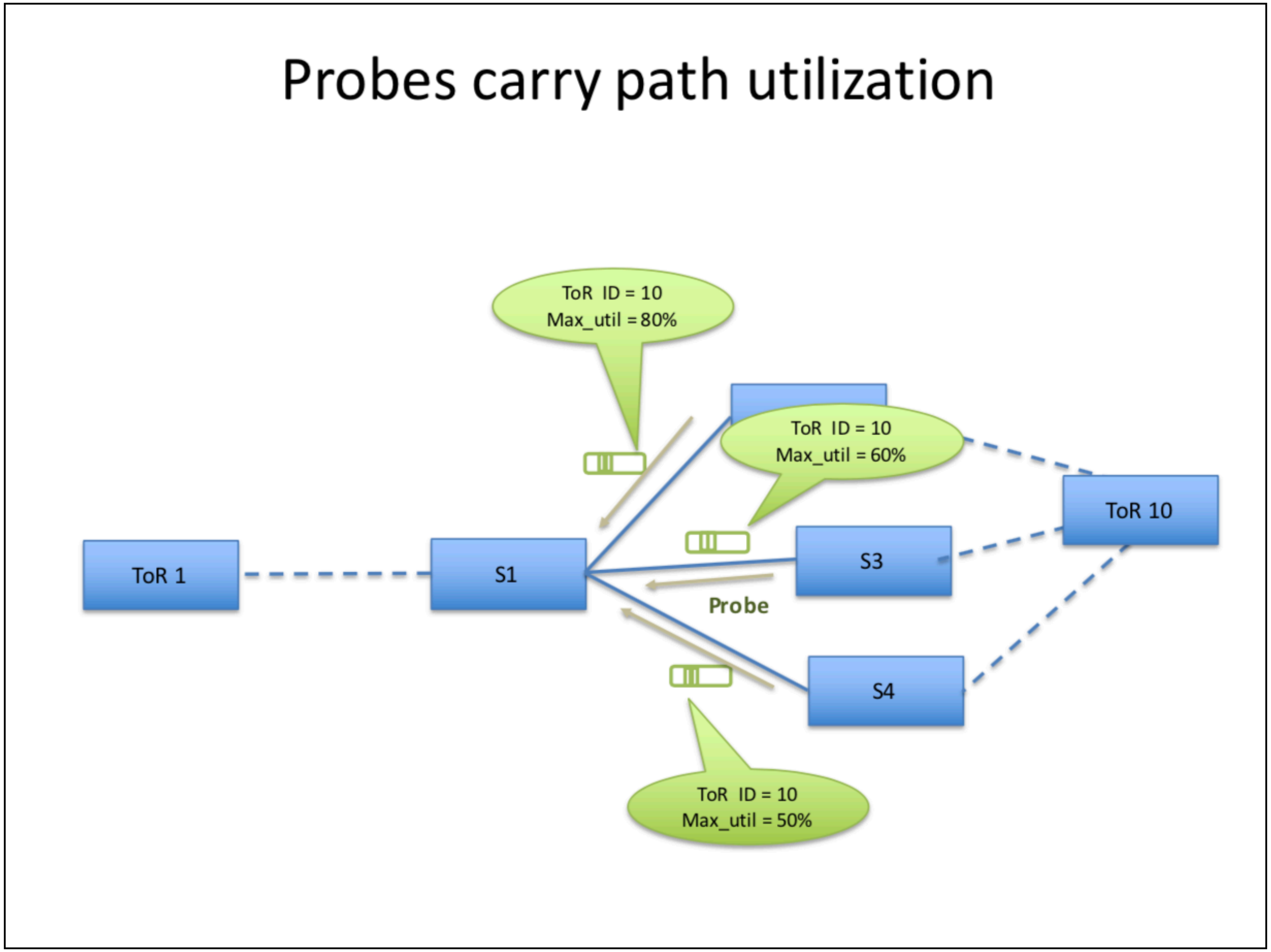
1. Periodic INT probes
   - disseminate path utilization to switches
2. Flowlet detection and path selection
   - happens at all switches
   - hop-by-hop adaptive routing

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

INT probes traverse multiple paths

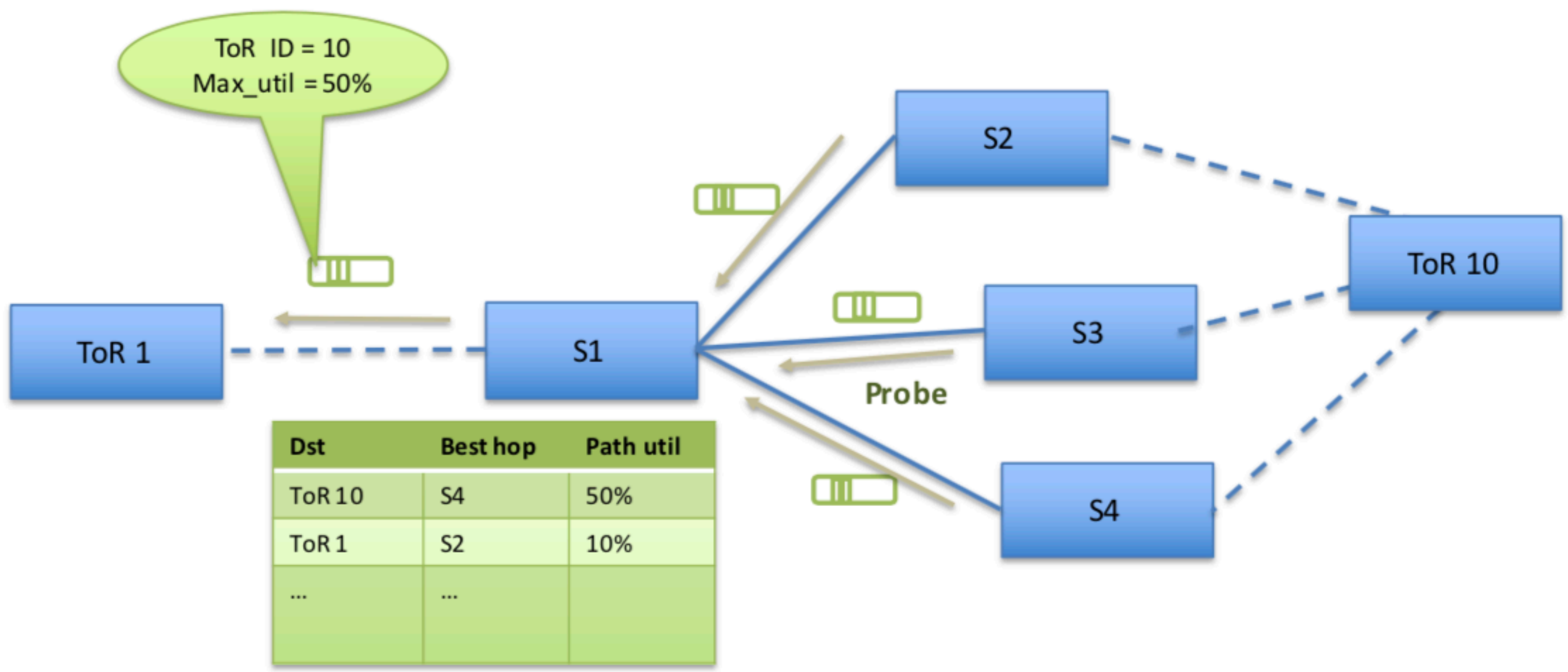


Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015



Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

Probes update switch state



Dst	Best hop	Path util
ToR 10	S4	50%
ToR 1	S2	10%
...	...	

Path Util table

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

Summary

- INT provides real-time network state directly in the dataplane
  - Scales to arbitrarily large networks
  - Scales to current and future link speeds
  - Can adapt to any network, any encap, any application
- Knowledge of real-time network state opens up new possibilities
  - Enhanced monitoring and troubleshooting
  - Network-state aware routing
  - …

Source: In-band Network Telemetry, Mukesh Hira and Naga Katta, 2015

Language-Directed Hardware Design for Network Performance Monitoring

Sonata: Query-Driven Streaming Network Telemetry

Arpit Gupta, Princeton University
Rob Harrison, Princeton University
Marco Canini, KAUST
Nick Feamster, Princeton University
Jennifer Rexford, Princeton University
Walter Willinger, NIKSUN Inc.

LossRadar: Fast Detection of Lost Packets in Data Center Networks

FlowRadar: A Better NetFlow for Data Centers

Yuliang Li, Rui Miao, Changhoon Kim, Minlan Yu
University of Southern California, Barefoot Networks

Dapper: Data Plane Performance Diagnosis of TCP

Network-Wide Heavy Hitter Detection with Commodity Switches

Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford
Princeton University

In-band Network Telemetry (INT)

June 2016

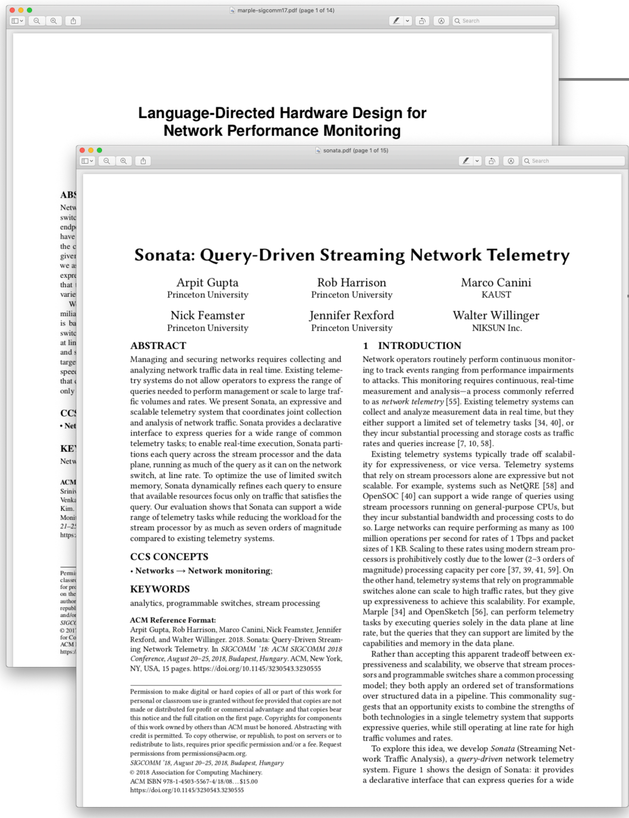
Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Elastic Sketch: Adaptive and Fast Network-wide Measurements

One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon

Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, Vladimir Braverman
Johns Hopkins University, Carnegie Mellon University



MARPLE [SIGCOMM'17]

SONATA [SIGCOMM'18]

Both papers enable operators to express monitoring queries

```
result = filter(pktstream, qid == Q and switch == S
                and t_out - t_in > 1ms)
```

returns a stream of packets experiencing high queuing latencies

A compiler then compiles these queries to: switch programs + control code

The two papers differ among others in the types of queries they support

LossRadar [CoNEXT'16]

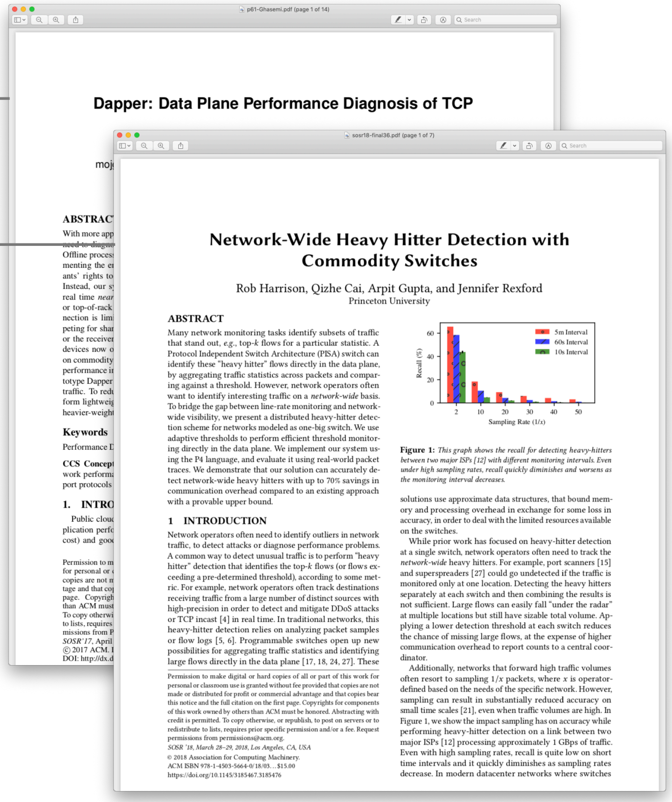
FlowRadar [NSDI'16]

Develop techniques and tools to monitor *all flows* by

- relying on in-switch data structures (Bloom Filters) and
- decoding them at the controller-level

DAPPER [SOSR'17]

Network-Wide HH [SOSR'18]



Dapper: Data Plane Performance Diagnosis of TCP

Network-Wide Heavy Hitter Detection with Commodity Switches

Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford
Princeton University

ABSTRACT

Many network monitoring tasks identify subsets of traffic that stand out, *e.g.*, top-*k* flows for a particular statistic. A Protocol Independent Switch Architecture (PISA) switch can identify these "heavy hitter" flows directly in the data plane, by aggregating traffic statistics across packets and comparing against a threshold. However, network operators often want to identify interesting traffic on a *network-wide* basis. To bridge the gap between line-rate monitoring and network-wide visibility, we present a distributed heavy-hitter detection scheme for networks modeled as one-big switch. We use adaptive thresholds to perform efficient threshold monitoring directly in the data plane. We implement our system using the P4 language, and evaluate it using real-world packet traces. We demonstrate that our solution can accurately detect network-wide heavy hitters with up to 70% savings in communication overhead compared to an existing approach with a provable upper bound.

1 INTRODUCTION

Network operators often need to identify outliers in network traffic, to detect attacks or diagnose performance problems. A common way to detect unusual traffic is to perform "heavy hitter" detection that identifies the top-*k* flows (or flows exceeding a pre-determined threshold), according to some metric. For example, network operators often track destinations receiving traffic from a large number of distinct sources with high-precision in order to detect and mitigate DDoS attacks or TCP incast [4] in real time. In traditional networks, this heavy-hitter detection relies on analyzing packet samples or flow logs [5, 6]. Programmable switches open up new possibilities for aggregating traffic statistics and identifying large flows directly in the data plane [17, 18, 24, 27]. These

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SOSR '18, March 28–29, 2018, Los Angeles, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185476

Recall (%)
5m Interval
60s Interval
10s Interval
Sampling Rate (1/x)

Figure 1: *This graph shows the recall for detecting heavy-hitters between two major ISPs [12] with different monitoring intervals. Even under high sampling rates, recall quickly diminishes and worsens as the monitoring interval decreases.*

solutions use approximate data structures, that bound memory and processing overhead in exchange for some loss in accuracy, in order to deal with the limited resources available on the switches.

While prior work has focused on heavy-hitter detection at a single switch, network operators often need to track the *network-wide* heavy hitters. For example, port scanners [15] and superspreaders [27] could go undetected if the traffic is monitored only at one location. Detecting the heavy hitters separately at each switch and then combining the results is not sufficient. Large flows can easily fall "under the radar" at multiple locations but still have sizable total volume. Applying a lower detection threshold at each switch reduces the chance of missing large flows, at the expense of higher communication overhead to report counts to a central coordinator.

Additionally, networks that forward high traffic volumes often resort to sampling $1/x$ packets, where x is operator-defined based on the needs of the specific network. However, sampling can result in substantially reduced accuracy on small time scales [21], even when traffic volumes are high. In Figure 1, we show the impact sampling has on accuracy while performing heavy-hitter detection on a link between two major ISPs [12] processing approximately 1 GBps of traffic. Even with high sampling rates, recall is quite low on short time intervals and it quickly diminishes as sampling rates decrease. In modern datacenter networks where switches

Develop P4-based detection mechanisms to

- diagnose TCP performance issue (e.g. small receiver buffers)
- heavy-hitter (e.g. port scanners, superspreader, DDoS)

Introduce techniques to make sketch-based monitoring more practical (by making sketches adaptive or "universal")

SketchLearn [SIGCOMM'18]

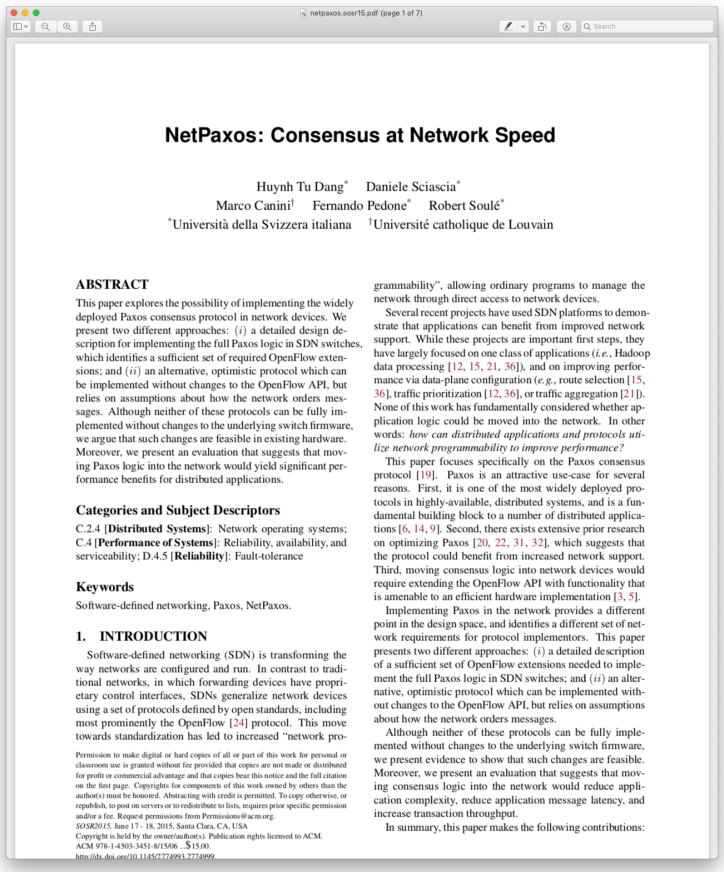
Elastic Sketch [SIGCOMM'18]

UnivMon [SIGCOMM'16]

Data plane programmability	for	Performance Monitoring Applications offloading

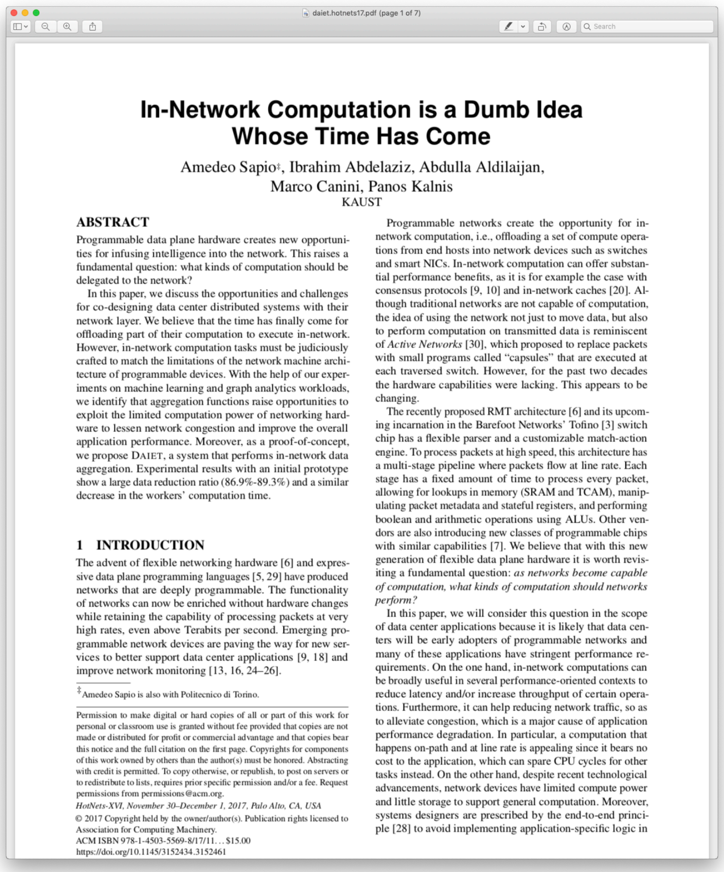
Platforms Correctness Management	for	Data plane programmability

[SOSR'15]



Consensus at network speed

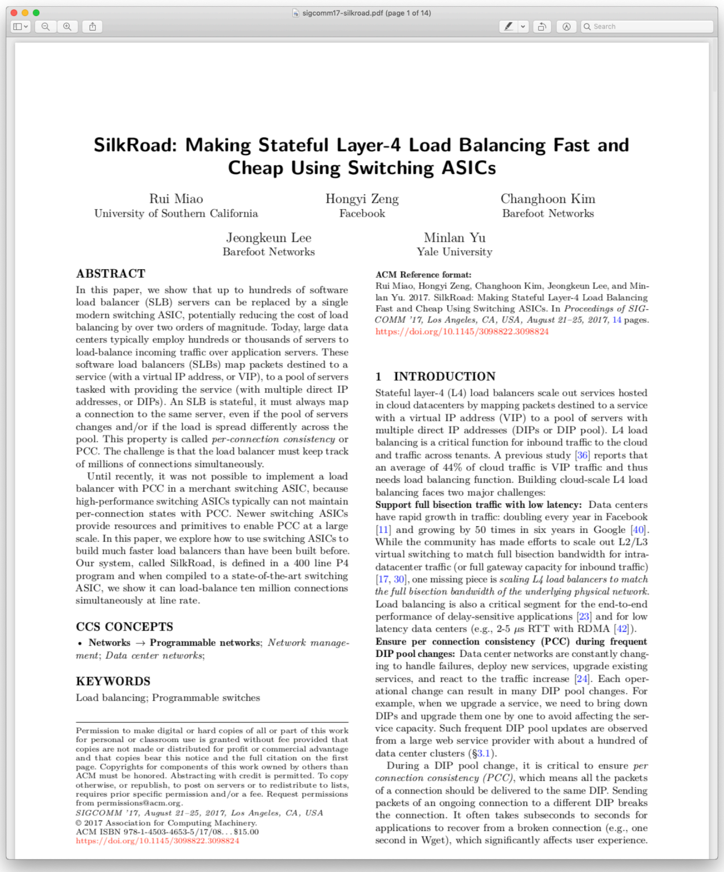
[HotNets'17]



In-Network Aggregation

(e.g., for MapReduce, graph analytics, ML)

[SIGCOMM'17]



Stateful layer-4 load balancers

+ NetCache [SOSP'17], NetChain [NSDI'18]

Data plane programmability for

Performance

Monitoring

Applications offloading

Platforms for Data plane programmability

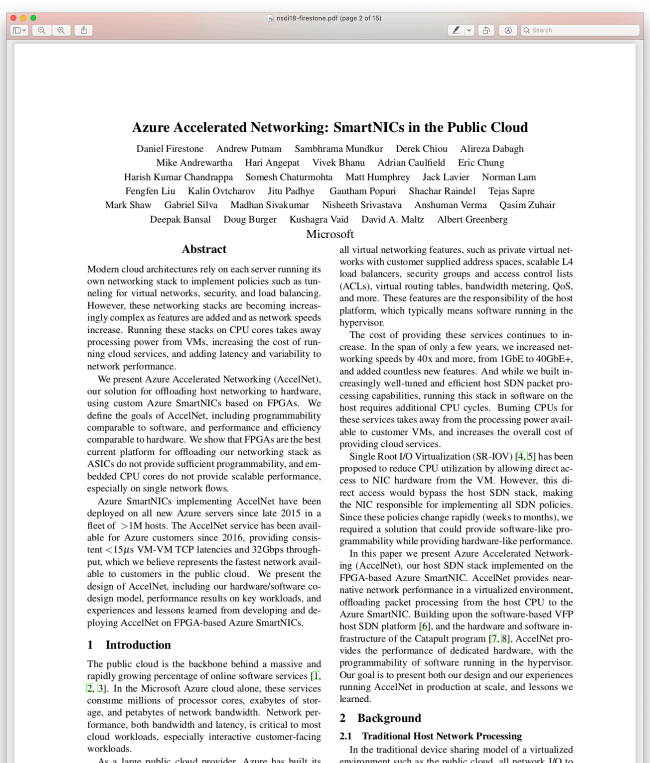
Correctness

Management

"Data-plane" programmability goes beyond switch programmability (or P4 for that matter)

Offloading… … to FPGA-based SmartNICS

host networking

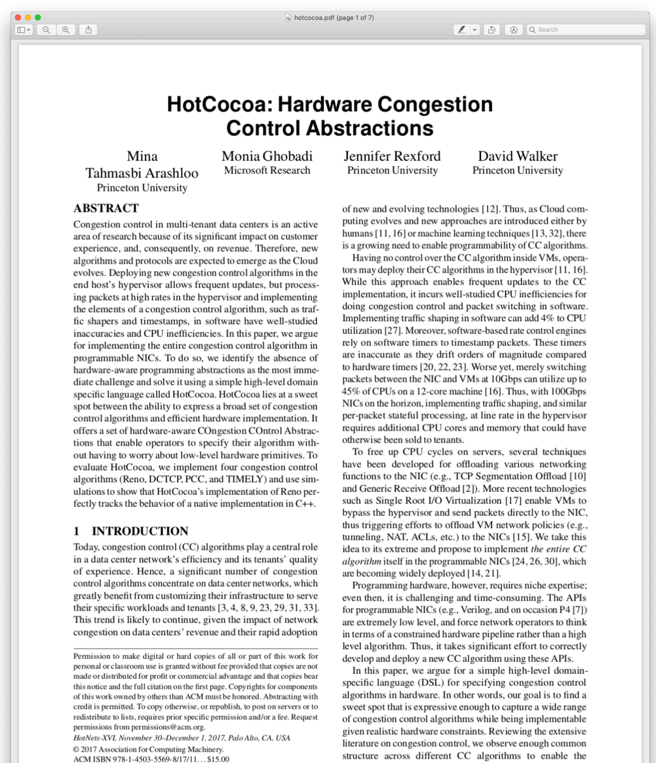

Azure Accelerated Networking: SmartNICs in the Public Cloud

Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, Albert Greenberg

Microsoft

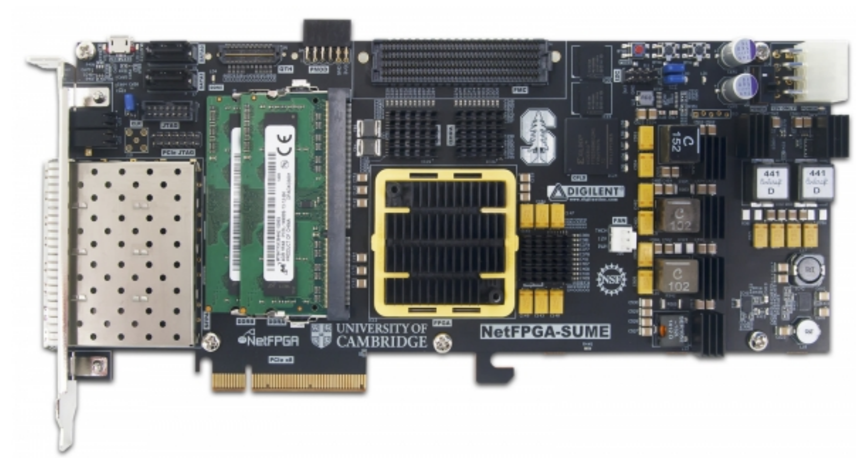
[NSDI'18]

congestion control


HotCocoa: Hardware Congestion Control Abstractions

Mina Tahmasbi Arashloo, Princeton University; Monia Ghobadi, Microsoft Research; Jennifer Rexford, Princeton University; David Walker, Princeton University

[HotNets'17]

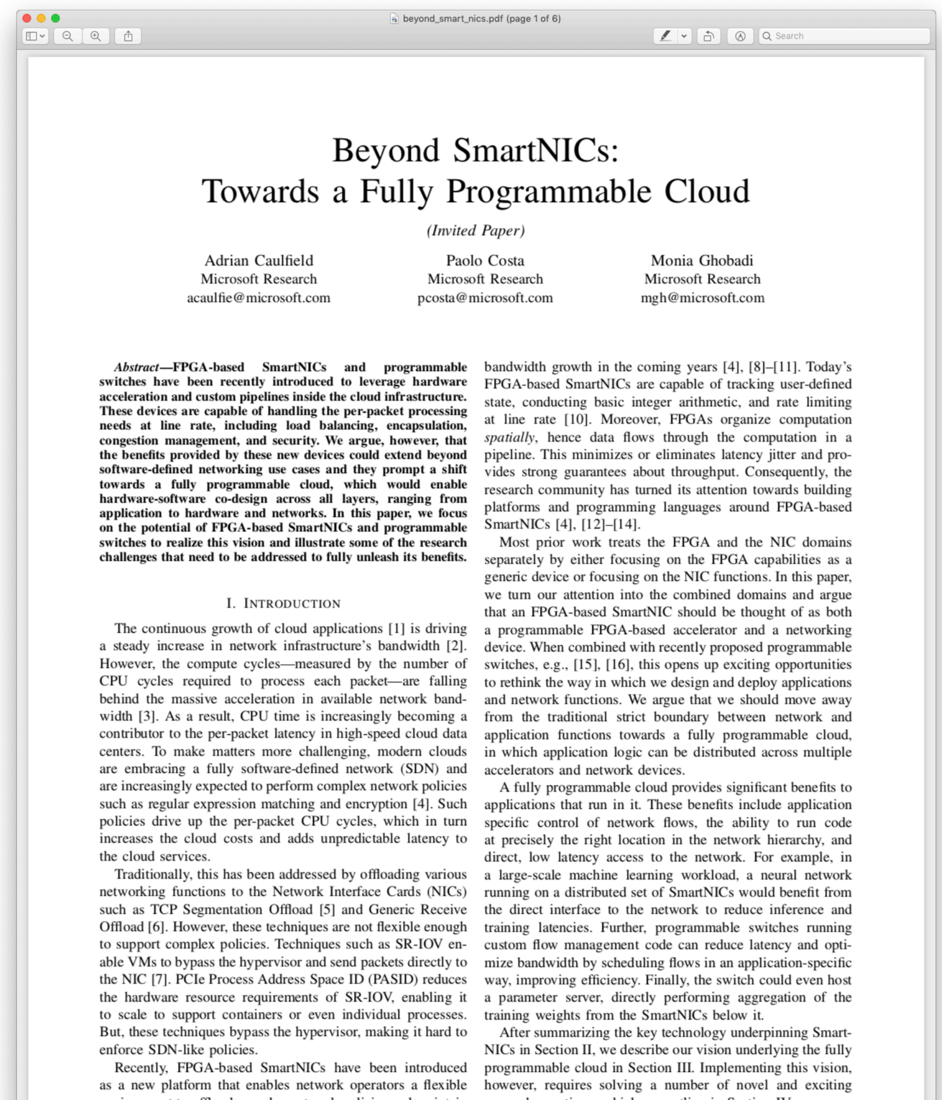


NetFPGA SUME board

Host-based programmability + SmartNICs + programmable switches = fully programmable platforms

Big question is

How to combine them best?



Beyond SmartNICs: Towards a Fully Programmable Cloud

(Invited Paper)

Adrian Caulfield, Microsoft Research, acaulfie@microsoft.com

Paolo Costa, Microsoft Research, pcosta@microsoft.com

Monia Ghobadi, Microsoft Research, mgh@microsoft.com

***Abstract*—FPGA-based SmartNICs and programmable switches have been recently introduced to leverage hardware acceleration and custom pipelines inside the cloud infrastructure. These devices are capable of handling the per-packet processing needs at line rate, including load balancing, encapsulation, congestion management, and security. We argue, however, that the benefits provided by these new devices could extend beyond software-defined networking use cases and they prompt a shift towards a fully programmable cloud, which would enable hardware-software co-design across all layers, ranging from application to hardware and networks. In this paper, we focus on the potential of FPGA-based SmartNICs and programmable switches to realize this vision and illustrate some of the research challenges that need to be addressed to fully unleash its benefits.**

I. Introduction

The continuous growth of cloud applications [1] is driving a steady increase in network infrastructure's bandwidth [2]. However, the compute cycles—measured by the number of CPU cycles required to process each packet—are falling behind the massive acceleration in available network bandwidth [3]. As a result, CPU time is increasingly becoming a contributor to the per-packet latency in high-speed cloud data centers. To make matters more challenging, modern clouds are embracing a fully software-defined network (SDN) and are increasingly expected to perform complex network policies such as regular expression matching and encryption [4]. Such policies drive up the per-packet CPU cycles, which in turn increases the cloud costs and adds unpredictable latency to the cloud services.

Traditionally, this has been addressed by offloading various networking functions to the Network Interface Cards (NICs) such as TCP Segmentation Offload [5] and Generic Receive Offload [6]. However, these techniques are not flexible enough to support complex policies. Techniques such as SR-IOV enable VMs to bypass the hypervisor and send packets directly to the NIC [7]. PCIe Process Address Space ID (PASID) reduces the hardware resource requirements of SR-IOV, enabling it to scale to support containers or even individual processes. But, these techniques bypass the hypervisor, making it hard to enforce SDN-like policies.

Recently, FPGA-based SmartNICs have been introduced as a new platform that enables network operators a flexible environment to offload complex network policies and maintain bandwidth growth in the coming years [4], [8]–[11]. Today's FPGA-based SmartNICs are capable of tracking user-defined state, conducting basic integer arithmetic, and rate limiting at line rate [10]. Moreover, FPGAs organize computation *spatially*, hence data flows through the computation in a pipeline. This minimizes or eliminates latency jitter and provides strong guarantees about throughput. Consequently, the research community has turned its attention towards building platforms and programming languages around FPGA-based SmartNICs [4], [12]–[14].

Most prior work treats the FPGA and the NIC domains separately by either focusing on the FPGA capabilities as a generic device or focusing on the NIC functions. In this paper, we turn our attention into the combined domains and argue that an FPGA-based SmartNIC should be thought of as both a programmable FPGA-based accelerator and a networking device. When combined with recently proposed programmable switches, e.g., [15], [16], this opens up exciting opportunities to rethink the way in which we design and deploy applications and network functions. We argue that we should move away from the traditional strict boundary between network and application functions towards a fully programmable cloud, in which application logic can be distributed across multiple accelerators and network devices.

A fully programmable cloud provides significant benefits to applications that run in it. These benefits include application specific control of network flows, the ability to run code at precisely the right location in the network hierarchy, and direct, low latency access to the network. For example, in a large-scale machine learning workload, a neural network running on a distributed set of SmartNICs would benefit from the direct interface to the network to reduce inference and training latencies. Further, programmable switches running custom flow management code can reduce latency and optimize bandwidth by scheduling flows in an application-specific way, improving efficiency. Finally, the switch could even host a parameter server, directly performing aggregation of the training weights from the SmartNICs below it.

After summarizing the key technology underpinning SmartNICs in Section II, we describe our vision underlying the fully programmable cloud in Section III. Implementing this vision, however, requires solving a number of novel and exciting research questions, which we outline in Section IV.

IEEE International Conference on High Performance Switching and Routing, 2018

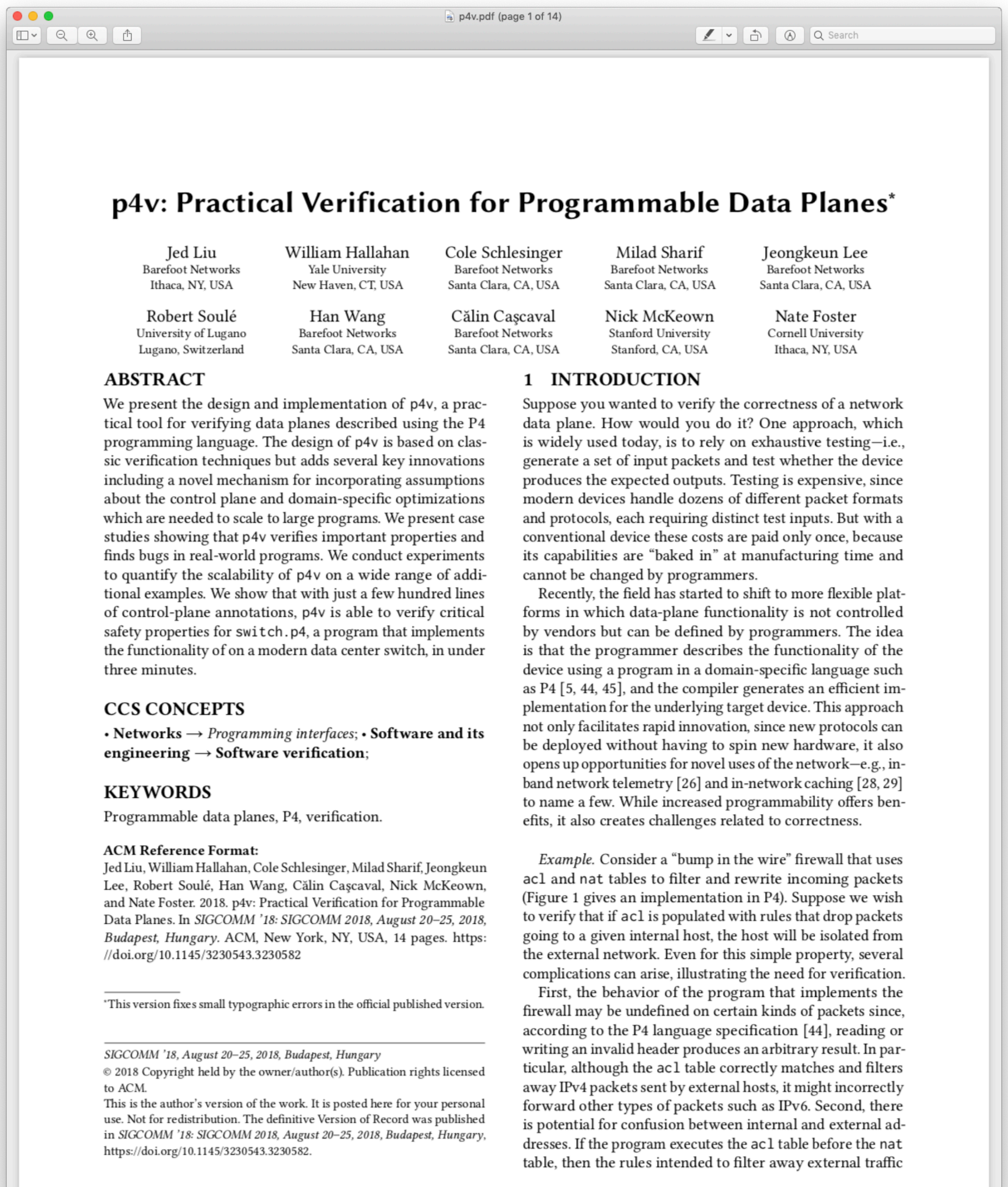
Data plane programmability for

Performance
Monitoring
Applications offloading

Platforms for Data plane programmability
Correctness
Management

So you've a programmable networks…

How do you make sure that it works as it should?!



p4v.pdf (page 1 of 14)

p4v: Practical Verification for Programmable Data Planes*

Jed Liu, Barefoot Networks, Ithaca, NY, USA
William Hallahan, Yale University, New Haven, CT, USA
Cole Schlesinger, Barefoot Networks, Santa Clara, CA, USA
Milad Sharif, Barefoot Networks, Santa Clara, CA, USA
Jeongkeun Lee, Barefoot Networks, Santa Clara, CA, USA
Robert Soulé, University of Lugano, Lugano, Switzerland
Han Wang, Barefoot Networks, Santa Clara, CA, USA
Călin Caşcaval, Barefoot Networks, Santa Clara, CA, USA
Nick McKeown, Stanford University, Stanford, CA, USA
Nate Foster, Cornell University, Ithaca, NY, USA

ABSTRACT

We present the design and implementation of p4v, a practical tool for verifying data planes described using the P4 programming language. The design of p4v is based on classic verification techniques but adds several key innovations including a novel mechanism for incorporating assumptions about the control plane and domain-specific optimizations which are needed to scale to large programs. We present case studies showing that p4v verifies important properties and finds bugs in real-world programs. We conduct experiments to quantify the scalability of p4v on a wide range of additional examples. We show that with just a few hundred lines of control-plane annotations, p4v is able to verify critical safety properties for switch.p4, a program that implements the functionality of on a modern data center switch, in under three minutes.

CCS CONCEPTS

• **Networks** → *Programming interfaces*; • **Software and its engineering** → **Software verification**;

KEYWORDS

Programmable data planes, P4, verification.

ACM Reference Format:
Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. 2018. p4v: Practical Verification for Programmable Data Planes. In *SIGCOMM '18: SIGCOMM 2018, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3230543.3230582

*This version fixes small typographic errors in the official published version.

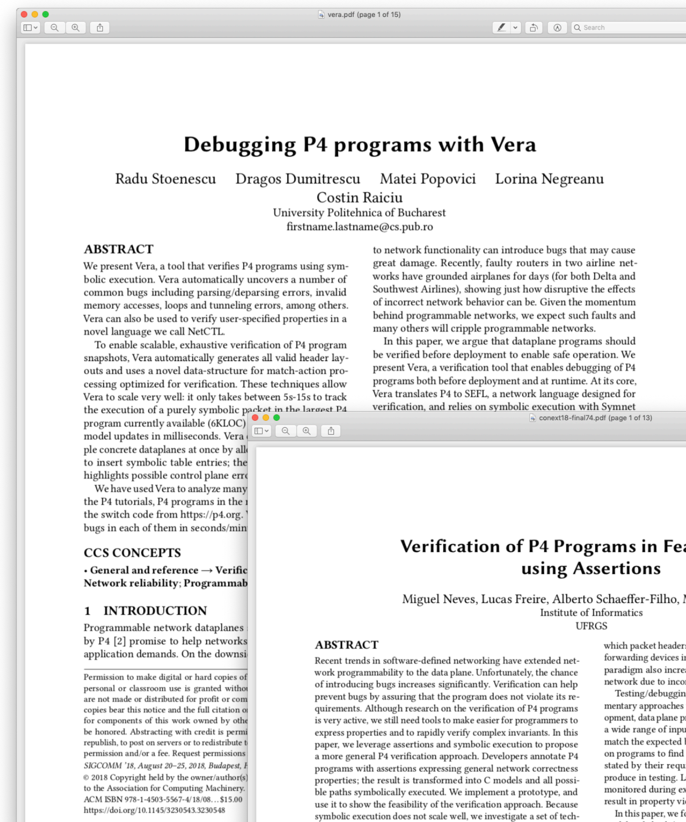
1 INTRODUCTION

Suppose you wanted to verify the correctness of a network data plane. How would you do it? One approach, which is widely used today, is to rely on exhaustive testing—i.e., generate a set of input packets and test whether the device produces the expected outputs. Testing is expensive, since modern devices handle dozens of different packet formats and protocols, each requiring distinct test inputs. But with a conventional device these costs are paid only once, because its capabilities are "baked in" at manufacturing time and cannot be changed by programmers.

Recently, the field has started to shift to more flexible platforms in which data-plane functionality is not controlled by vendors but can be defined by programmers. The idea is that the programmer describes the functionality of the device using a program in a domain-specific language such as P4 [5, 44, 45], and the compiler generates an efficient implementation for the underlying target device. This approach not only facilitates rapid innovation, since new protocols can be deployed without having to spin new hardware, it also opens up opportunities for novel uses of the network—e.g., inband network telemetry [26] and in-network caching [28, 29] to name a few. While increased programmability offers benefits, it also creates challenges related to correctness.

Example. Consider a "bump in the wire" firewall that uses acl and nat tables to filter and rewrite incoming packets (Figure 1 gives an implementation in P4). Suppose we wish to verify that if acl is populated with rules that drop packets going to a given internal host, the host will be isolated from the external network. Even for this simple property, several complications can arise, illustrating the need for verification.

First, the behavior of the program that implements the firewall may be undefined on certain kinds of packets since, according to the P4 language specification [44], reading or writing an invalid header produces an arbitrary result. In particular, although the acl table correctly matches and filters away IPv4 packets sent by external hosts, it might incorrectly forward other types of packets such as IPv6. Second, there is potential for confusion between internal and external addresses. If the program executes the acl table before the nat table, then the rules intended to filter away external traffic



vera.pdf (page 1 of 15)

Debugging P4 programs with Vera

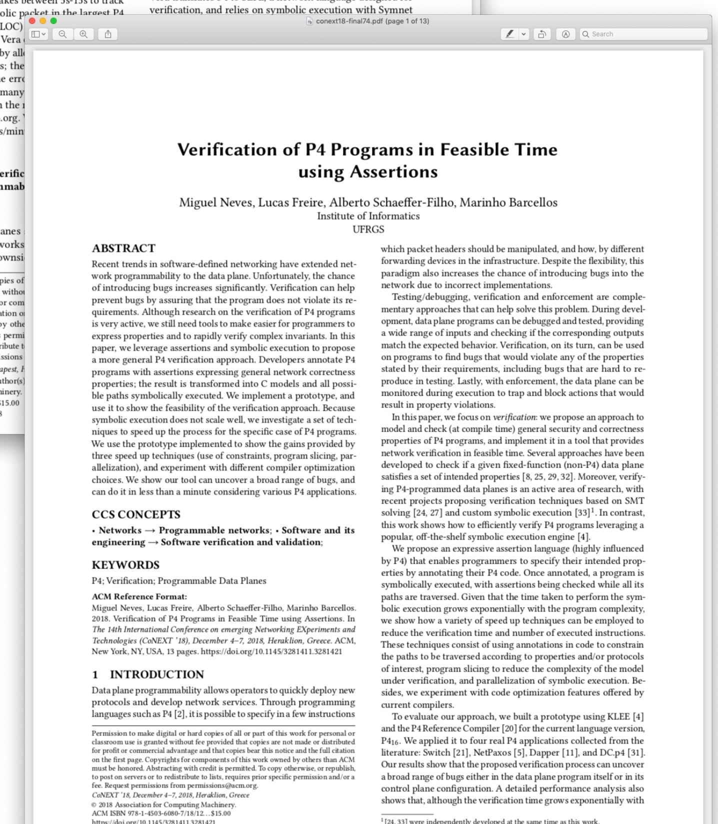
Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, Costin Raiciu
University Politehnica of Bucharest
firstname.lastname@cs.pub.ro

ABSTRACT

CCS CONCEPTS

1 INTRODUCTION

[SIGCOMM'18]



conext18-final74.pdf (page 1 of 13)

Verification of P4 Programs in Feasible Time using Assertions

Miguel Neves, Lucas Freire, Alberto Schaeffer-Filho, Marinho Barcellos
Institute of Informatics
UFRGS

ABSTRACT

CCS CONCEPTS

KEYWORDS

P4; Verification; Programmable Data Planes

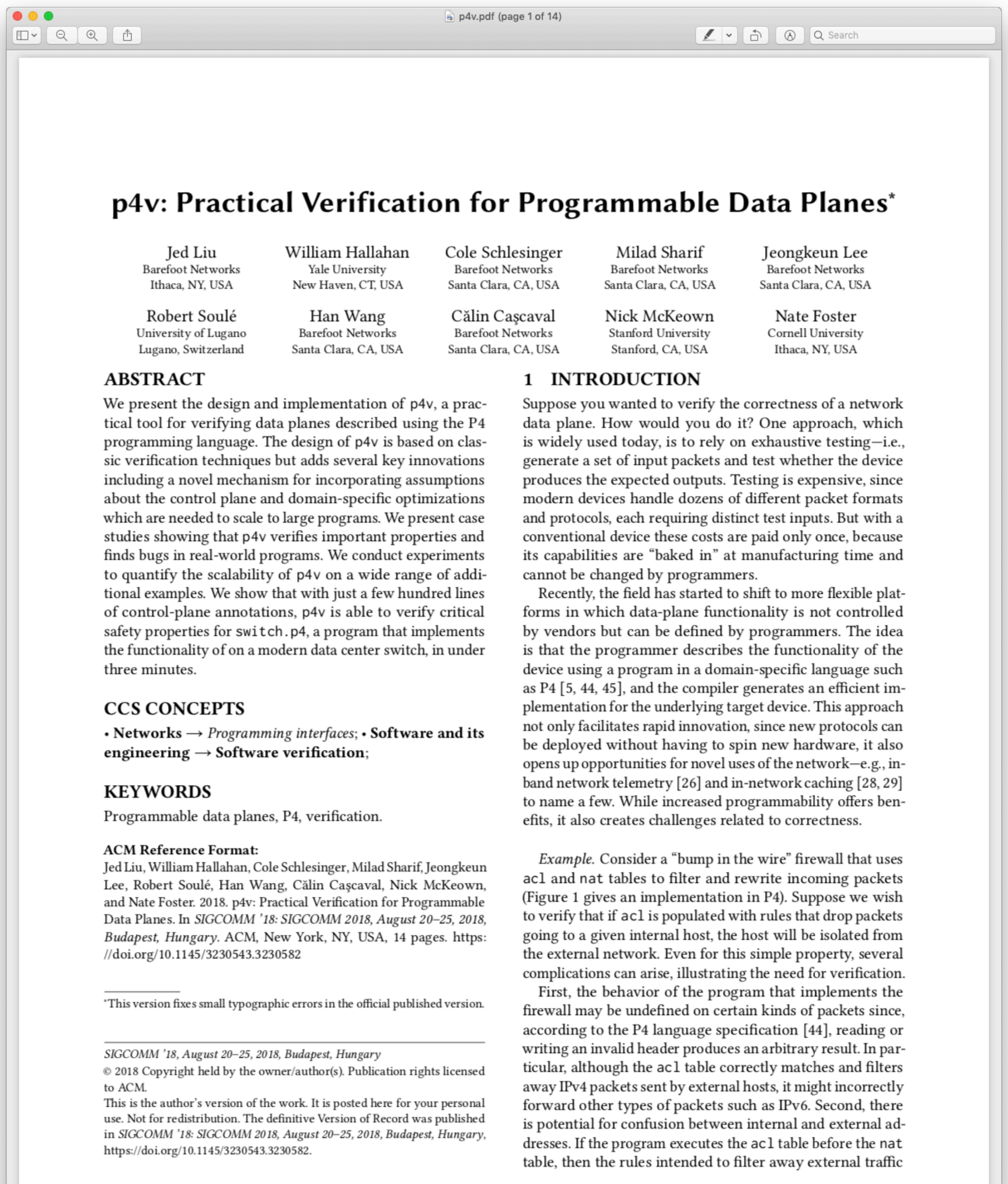
1 INTRODUCTION

[SIGCOMM'18]

[CoNEXT'18]

So you've a programmable networks…

How do you make sure that it works as it should?!



p4v.pdf (page 1 of 14)

p4v: Practical Verification for Programmable Data Planes*

Jed Liu, Barefoot Networks, Ithaca, NY, USA
William Hallahan, Yale University, New Haven, CT, USA
Cole Schlesinger, Barefoot Networks, Santa Clara, CA, USA
Milad Sharif, Barefoot Networks, Santa Clara, CA, USA
Jeongkeun Lee, Barefoot Networks, Santa Clara, CA, USA
Robert Soulé, University of Lugano, Lugano, Switzerland
Han Wang, Barefoot Networks, Santa Clara, CA, USA
Călin Caşcaval, Barefoot Networks, Santa Clara, CA, USA
Nick McKeown, Stanford University, Stanford, CA, USA
Nate Foster, Cornell University, Ithaca, NY, USA

ABSTRACT

We present the design and implementation of p4v, a practical tool for verifying data planes described using the P4 programming language. The design of p4v is based on classic verification techniques but adds several key innovations including a novel mechanism for incorporating assumptions about the control plane and domain-specific optimizations which are needed to scale to large programs. We present case studies showing that p4v verifies important properties and finds bugs in real-world programs. We conduct experiments to quantify the scalability of p4v on a wide range of additional examples. We show that with just a few hundred lines of control-plane annotations, p4v is able to verify critical safety properties for switch.p4, a program that implements the functionality of on a modern data center switch, in under three minutes.

CCS CONCEPTS

• Networks → *Programming interfaces*; • Software and its engineering → Software verification;

KEYWORDS

Programmable data planes, P4, verification.

ACM Reference Format:
Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. 2018. p4v: Practical Verification for Programmable Data Planes. In *SIGCOMM '18: SIGCOMM 2018, August 20–25, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3230543.3230582

*This version fixes small typographic errors in the official published version.

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *SIGCOMM '18: SIGCOMM 2018, August 20–25, 2018, Budapest, Hungary*, https://doi.org/10.1145/3230543.3230582.

1 INTRODUCTION

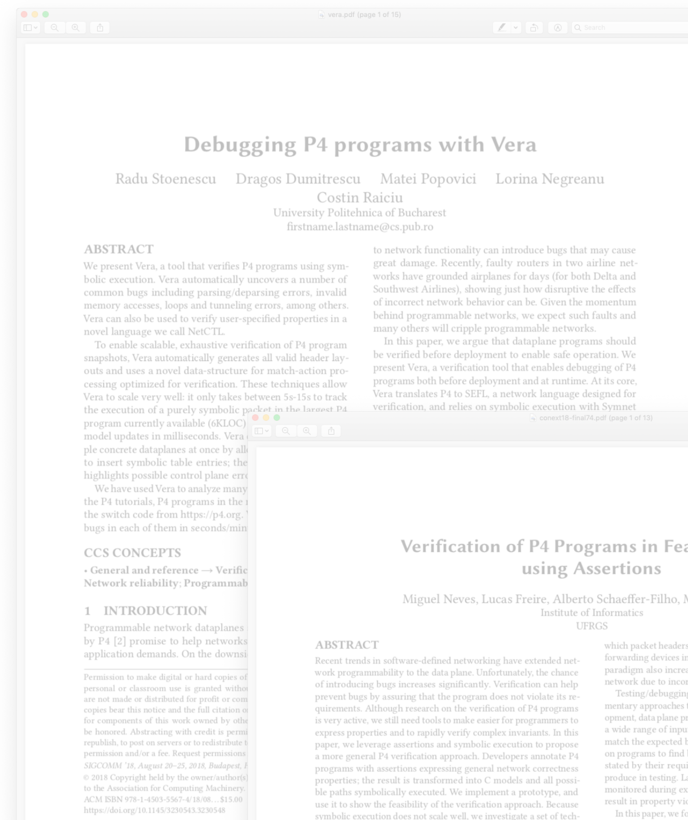
Suppose you wanted to verify the correctness of a network data plane. How would you do it? One approach, which is widely used today, is to rely on exhaustive testing—i.e., generate a set of input packets and test whether the device produces the expected outputs. Testing is expensive, since modern devices handle dozens of different packet formats and protocols, each requiring distinct test inputs. But with a conventional device these costs are paid only once, because its capabilities are "baked in" at manufacturing time and cannot be changed by programmers.

Recently, the field has started to shift to more flexible platforms in which data-plane functionality is not controlled by vendors but can be defined by programmers. The idea is that the programmer describes the functionality of the device using a program in a domain-specific language such as P4 [5, 44, 45], and the compiler generates an efficient implementation for the underlying target device. This approach not only facilitates rapid innovation, since new protocols can be deployed without having to spin new hardware, it also opens up opportunities for novel uses of the network—e.g., in-band network telemetry [26] and in-network caching [28, 29] to name a few. While increased programmability offers benefits, it also creates challenges related to correctness.

Example. Consider a "bump in the wire" firewall that uses acl and nat tables to filter and rewrite incoming packets (Figure 1 gives an implementation in P4). Suppose we wish to verify that if acl is populated with rules that drop packets going to a given internal host, the host will be isolated from the external network. Even for this simple property, several complications can arise, illustrating the need for verification.

First, the behavior of the program that implements the firewall may be undefined on certain kinds of packets since, according to the P4 language specification [44], reading or writing an invalid header produces an arbitrary result. In particular, although the acl table correctly matches and filters away IPv4 packets sent by external hosts, it might incorrectly forward other types of packets such as IPv6. Second, there is potential for confusion between internal and external addresses. If the program executes the acl table before the nat table, then the rules intended to filter away external traffic

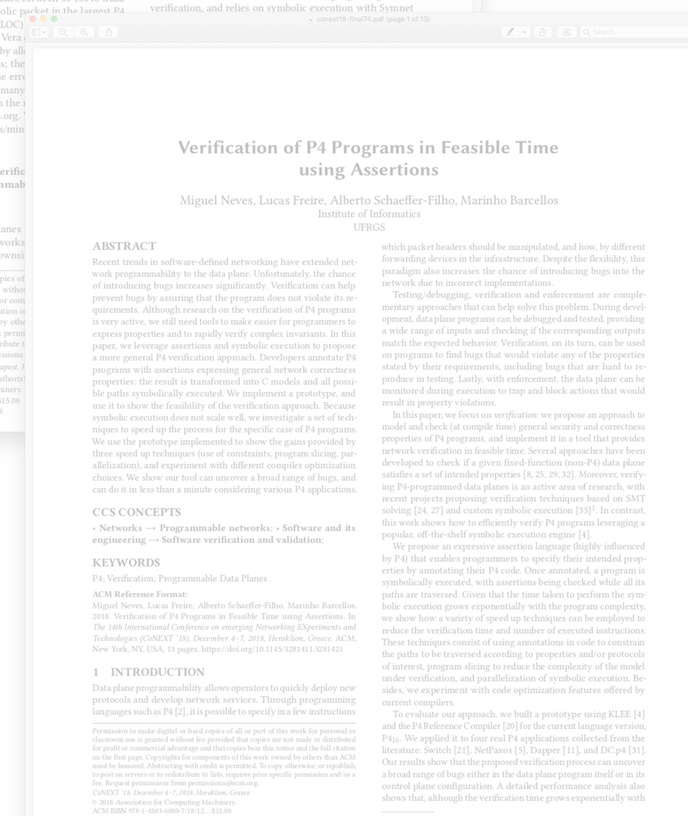
[SIGCOMM'18]



Debugging P4 programs with Vera

Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, Costin Raiciu
University Politehnica of Bucharest
firstname.lastname@cs.pub.ro

[SIGCOMM'18]



Verification of P4 Programs in Feasible Time using Assertions

Miguel Neves, Lucas Freire, Alberto Schaeffer-Filho, Marinho Barcellos
Institute of Informatics
UFRGS

[CoNEXT'18]

Programmable routers...

(specifically, programmable data planes)

...how do they work?

Arista 7170 series switches

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

Let's verify!

Bit-level description
of data-plane behaviour

Arista 7170 series switches

Give programmers language-based verification tools

P4 also used as HDL for fixed-function devices

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

P4 by example

- P4 is a low-level language → many gotchas
- Let's explore by example!
  - IPv6 router w/ access control list (ACL)

```
control ingress { apply(acl); }

table acl {
  reads { ipv6.dstAddr: lpm; }
  actions { allow; deny; }
}

action allow() {
  modify_field(std_meta.egress_spec, 1);
}

action deny() { drop(); }
```

What could *possibly* go wrong?

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

What if we didn't receive an IPv6 packet?

`ipv6` header will be **invalid**

What goes wrong

Table reads arbitrary values

→ Intended ACL policy violated

Can read values from a previous packet

→ Side channel vulnerability!

Real programs are complicated:
hard to keep validity in your head

```
control ingress { apply(acl); }

table acl {
  reads { ipv6.dstAddr: lpm; }
  actions { allow; deny; }
}

action allow() {
  modify_field(std_meta.egress_spec, 1);
}

action deny() { drop(); }
```

Property #1: header validity

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

What if `acl` table misses (no rule matches)?

Forwarding decision is unspecified

What goes wrong

Forwarding behaviour depends on hardware

- May not do what you expect!
- Code not portable

```
control ingress { apply(acl); }

table acl {
  reads { ipv6.dstAddr: lpm; }
  actions { allow; deny; }
}

action allow() {
  modify_field(std_meta.egress_spec, 1);
}

action deny() { drop(); }
```

Property #2: unambiguous forwarding

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

Types of properties

General safety

- **Header validity**
- Arithmetic-overflow checking
- Index bounds checking (header stacks, registers, meters, ...)

Architectural

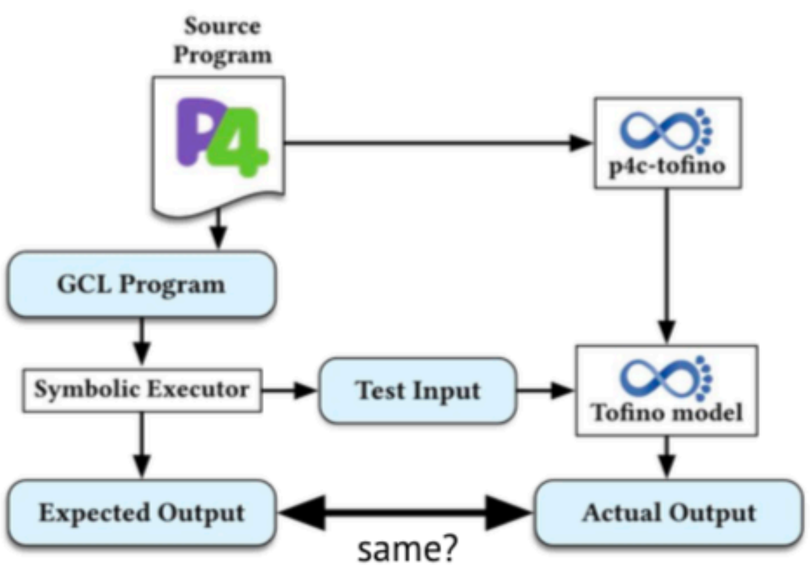
- **Unambiguous forwarding**
- **Reparseability**
- **Mutual exclusion of headers**
- Correct metadata usage (e.g., read-only metadata)

Program-specific

- Custom assertions in P4 program — e.g., IPv4 `ttl` correctly decremented

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

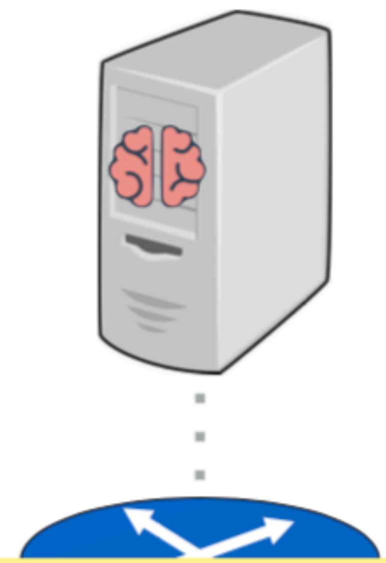
Challenge #1: imprecise semantics



- P4 language spec doesn't give precise semantics
- Defined semantics by translation to GCL (a simple imperative language)
- Tested semantics
  - Symbolically executed GCL to generate input–output tests for several programs
  - Ran w/ Barefoot P4 compiler & Tofino simulator

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

Challenge #2: modelling the control plane



- A P4 program is just half the program
  - Table rules are not statically known
  - Populated by the control plane at run time
- Control planes are carefully programmed
  - Tables rarely take arbitrary actions
- To rule out false positives, need to model behaviour of control plane

```
table acl {
  reads {
    ipv6.dstAddr: lpm;
  }
  actions { allow; deny; }
}
```

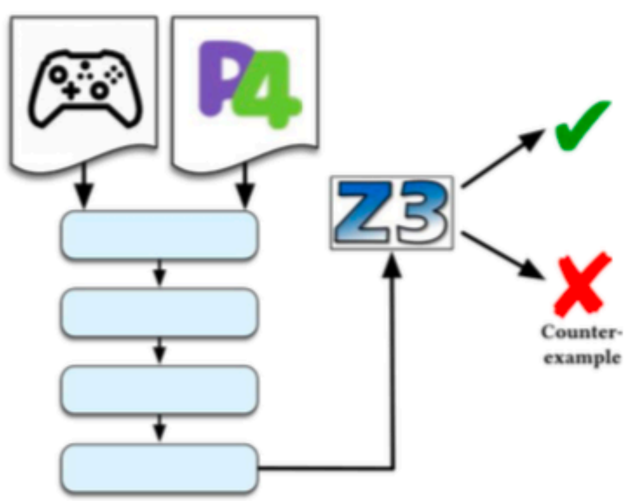
```
   ( @[ Action ] acl <hit> (allow);
     std_meta.egress_spec := 1)

[] ( @[ Action ] acl <hit> (deny);
     std_meta.egress_spec := 511)

[]   @[ Action ] acl <miss>
```

Tables translated into *unconstrained* nondeterministic choice

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

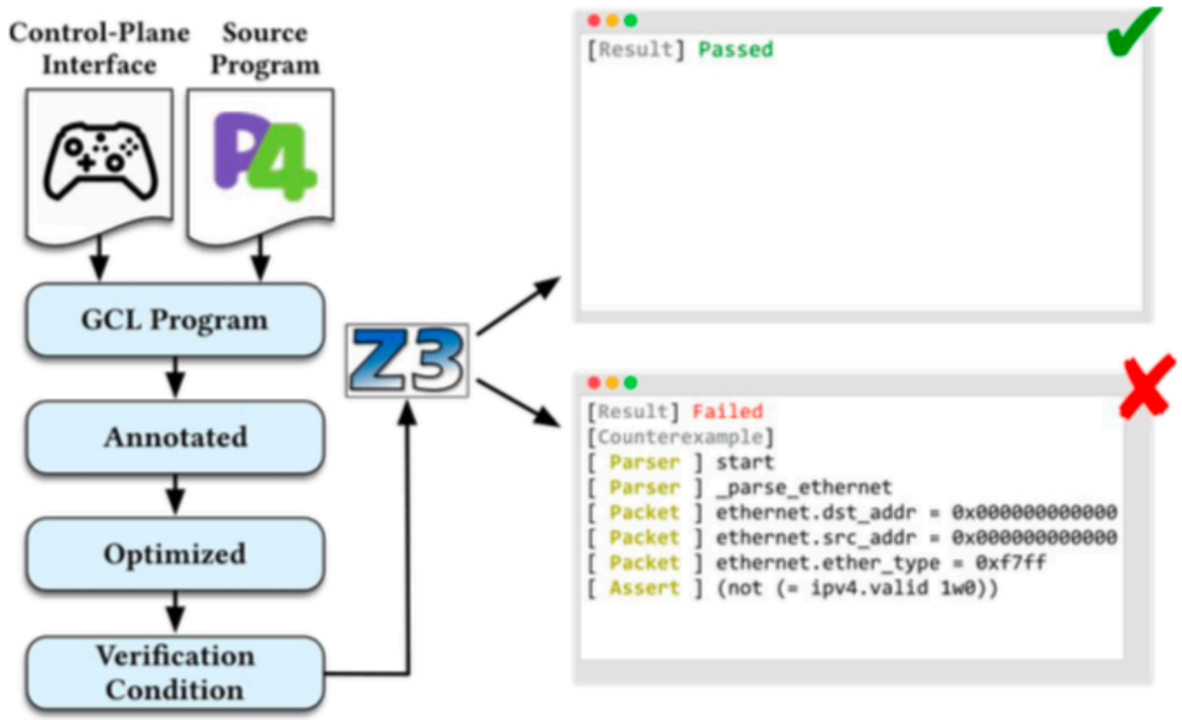
p4v overview

- **Automated** tool for verifying P4 programs
- Considers **all paths**
  - But also practical for **large programs**
- Includes basic safety properties for any program
- **Extensible** framework
  - Verify custom, program-specific properties
  - Assert-style debugging



Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

p4v architecture



1. Start w/ CPI & P4 program
2. Translate to GCL
3. Auto-annotate w/ assertions
4. Standard optimizations
5. Generate formula
6. Send to Z3
7. Success or counterexample
   - Input packet
   - Program trace
   - Violated assertion

Source: p4v, Practical Verification for Programmable Data Planes, Liu et al., 2018

Data plane programmability for

Performance

Monitoring

Applications offloading

Platforms for Data plane programmability

Correctness

Management

So you've a *verified* programmable networks…

How do you manage it?!

How do you perform planned maintenance?

now that you've state in your switches…

How do you run multiple applications in your switches?

monitoring, forwarding, load-balancing, etc.

How do you share resources amongst applications?

especially memory and # packet operations

We need an Operating System for the data plane

Definition
Wikipedia

An **operating system** is a system software that **manages** computer **hardware and software resources** and **provides common services** for computer programs.

Do we have that? Nope. Not yet at least.

We're working on it...

[SOSR'17]

vanbever_swing_state_sosr_2017.pdf (page 1 of 7)

Swing State: Consistent Updates for Stateful and Programmable Data Planes

Shouxi Luo* Hongfang Yu
University of Electronic Science and Technology of China

Laurent Vanbever
ETH Zürich

ABSTRACT

With the rise of stateful programmable data planes, a lot of the network functions that used to be implemented in the controller or at the end-hosts are now moving to the data plane to benefit from line-rate processing. Unfortunately, stateful data planes also mean more complex network updates as not only flows, but also the associated states, must now be migrated consistently to guarantee correct network behaviors. The main challenge is that data-plane states are maintained at line rate, according to possibly runtime criteria, rendering controller-driven migration impossible.

We present Swing State, a general state-management framework and runtime system supporting consistent state migration in stateful data planes. The key insight behind Swing State is to perform state migration entirely within the data plane by piggybacking state updates on live traffic. To minimize the overhead, Swing State only migrates the states that cannot be safely reconstructed at the destination switch.

We implemented a prototype of Swing State for P4. Given a P4 program, Swing State performs static analysis to compute which states require consistent migration and automatically augments the program to enable the transfer of these states at runtime. Our preliminary results indicate that Swing State is practical in migrating data-plane states at line rate with small overhead.

CCS Concepts

Keywords

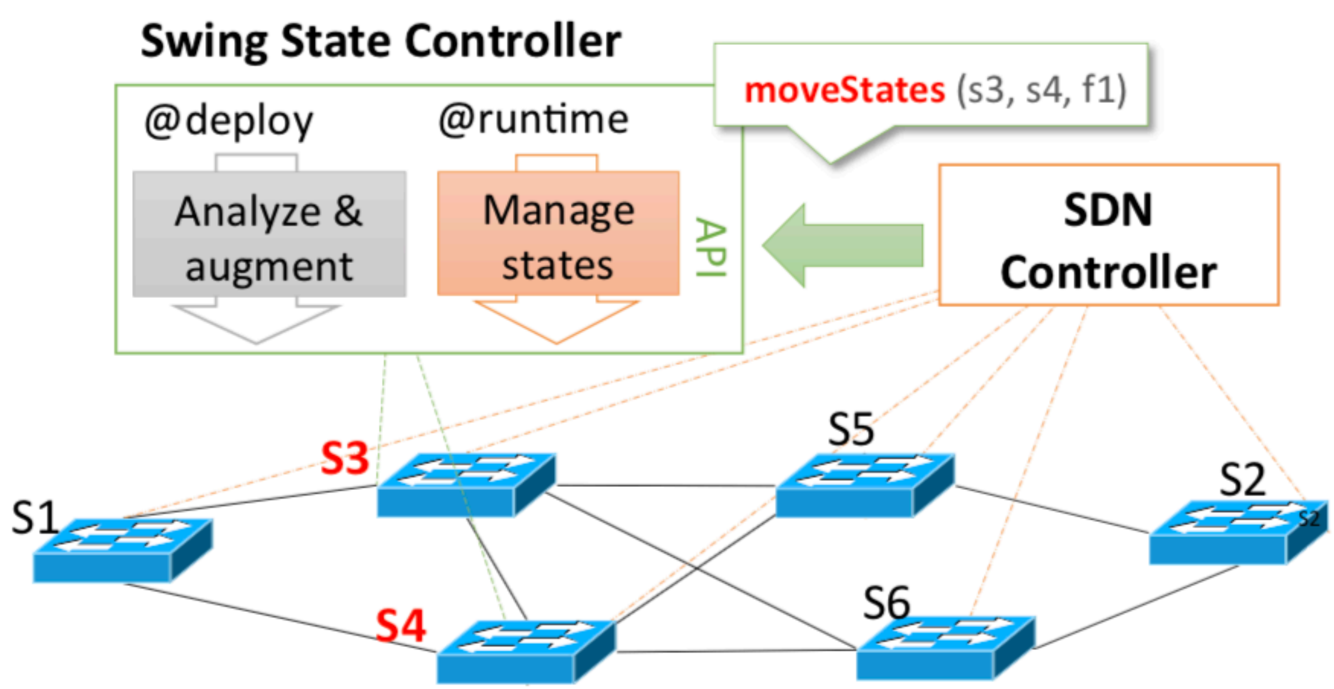
Network updates; Software-Defined Networking; P4; Stateful programmable data planes.

1. INTRODUCTION

By enabling stateful applications to run directly *in* the data plane, at line rate, programmable data planes [9, 22, 8, 28, 27, 16, 23] have recently emerged as a promising research area.

Yet, despite making SDNs more powerful, maintaining states in the data plane also calls for new consistent update mechanisms as it prevents traditional update techniques from working, and this, for three main reasons. First, the fact that data-plane states can be updated at line rate—at speeds that can reach Tbps [5]—prevents any software-based controller from consistently moving states from one device to another. Inconsistent migration is a problem for any data-plane application that requires strong-consistency network-wide. Examples of such applications include stateful firewalls tracking dynamic flow characteristics (e.g., low-level TCP states [29]) or anomaly detection applications [21]. Second, even ignoring states dynamism, the exact set of states to be migrated may actually be unknown to the controller, preventing it from performing the migration in the first place. Indeed, the states location in memory can differ from device to device according to runtime factors (e.g. a hash computed on packet headers) that are invisible to the controller. Third, data-plane states

Swing State is a state management framework with 1 primitive: moveStates



Source: Swing State: Consistent Updates for Stateful and Programmable Data Planes
Luo et al., SOSR 2017

Advanced Topics in Communication Networks

Lectures/Exercices

~7 weeks

how to program in P4

Group project

>= 7 weeks

in teams of 2—3

Advanced Topics in Communication Networks

Lectures/Exercices

~7 weeks

how to program in P4

Group project

>= 7 weeks

in teams of 2—3

The group project starts this week

It accounts for 50% of your final grade

The evaluation of your project will depend on your implementation, report, and presentation

The evaluation of your project will depend on your implementation, report, and presentation

implementation 70%	achieves the basic goals is properly documented runs…

The evaluation of your project will depend on your implementation, report, and presentation

implementation 70%	achieves the basic goals is properly documented runs…
report 15%, 10 pages max	describes the main building blocks evaluates the solution describes what each group member did

The evaluation of your project will depend on your implementation, report, and presentation

implementation 70%	achieves the basic goals is properly documented runs…
report 15%, 10 pages max	describes the main building blocks evaluates the solution describes what each group member did
presentation 15%, 12 min. +questions	summarizes the problem and the solution contains a *live* demo involves all group members

The final deadline for the project is

Wed Dec 19 at 23.59pm

This week	Select a proposal from the list (see Doodle) or send us your own proposal by email
Every week	Meet with the responsible assistant schedule a recurring slot in [10.15am; noon]
Wed Dec 19 11.59pm	Send us an archive with report, code, slides
Thu Dec 20 8.15am—	Groups presentation + course/exam debrief attendance is mandatory

The project has to be done in groups of 3 students

"Matching" process for incomplete groups via Slack

Project grade is shared by each group member provided that each collaborated (roughly equally)

- Let us know in advance if that's *not* the case
- Briefly describe in the report the contribution of each group member
- Each group member should be involved in the presentation and be able to answer questions

Details about each proposal is available on our website

Advanced Topics in Communication Networks
Project Proposals

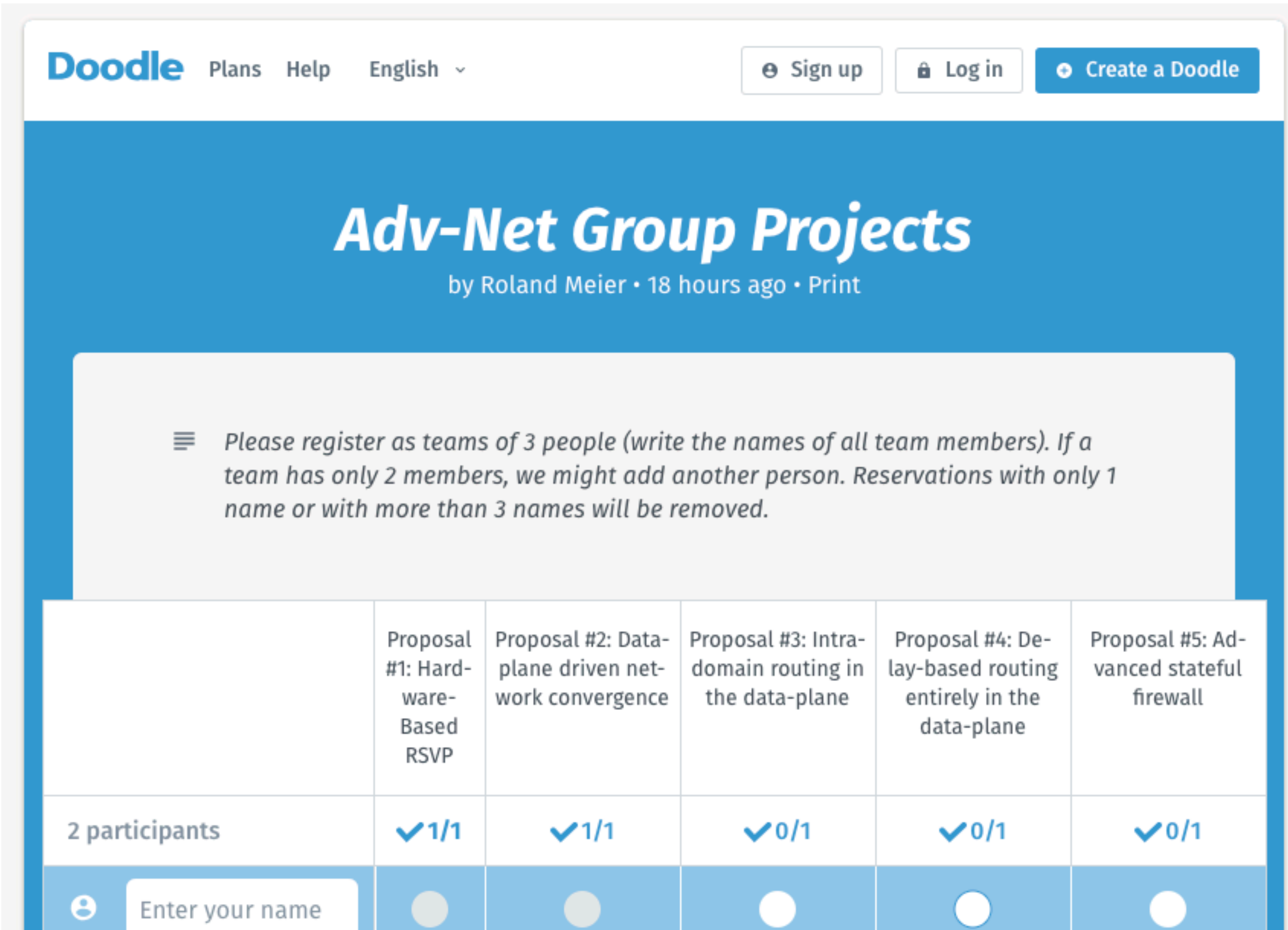
Proposal #1: Hardware-Based RSVP
Responsible: Albert Gran Alcoz

Resource Reservation Protocol (RSVP) [1] is a signaling protocol that allows connections in a network to perform bandwidth requests throughout a given path. It is a protocol that has been included in different solutions both in the traffic engineering field and in quality of service. Integrated Services (IntServ) was the first in adopting it, in the late 1990s, as a means to provide guaranteed quality of service in multimedia networks. Some years later, and with higher success, RSVP was extended for traffic engineering purposes in the RSVP-TE protocol [2] to be used for the establishment of virtual circuits in MPLS. RSVP suggests users in a network to perform bandwidth reservations before starting data transmissions. For that, packet probes are forwarded from source to destination, letting routers in between identify the amount of resources requested by the new connection. Routers will receive those requests and reply to them by annotating in the same packet their resource availability. Flows will only be admitted if all routers along the path have agreed on having enough resources for hosting the new request. Although achieving notable and robust performance, being able to provide 100% resource guarantees, the high price that RSVP requires in terms of scalability and complexity, has made from it a not very successful solution in multiple scenarios until nowadays. Among the main drawbacks, the most remarkable ones are the time required to set up a new connection (too high especially for real-time flows), the amount of state to be stored in each switch along the path (to keep track of reservations), and the periodic overheads needed to refresh reservation requests.

In this project, we propose the design and implementation of an evolved version of RSVP, based on P4, to be run directly on hardware. We strongly believe that a signaling protocol executed at line rate in the data-plane can be quicker in deploying configurations and faster in reacting to updates.

Students are expected to come up with a data-plane implementation, aiming to overcome RSVP original

Register your proposal (one per group)
from Friday 3pm until Sunday 11.59pm



If you want to propose your own project,
send me an email describing it by Friday (Nov 2) 3pm

lvanbever@ethz.ch

Quick overview of the proposals

Albert

Thomas

Roland

Alexander

Edgar

Quick overview of the proposals

Albert

Thomas

Roland

Alexander

Edgar

Proposal #1

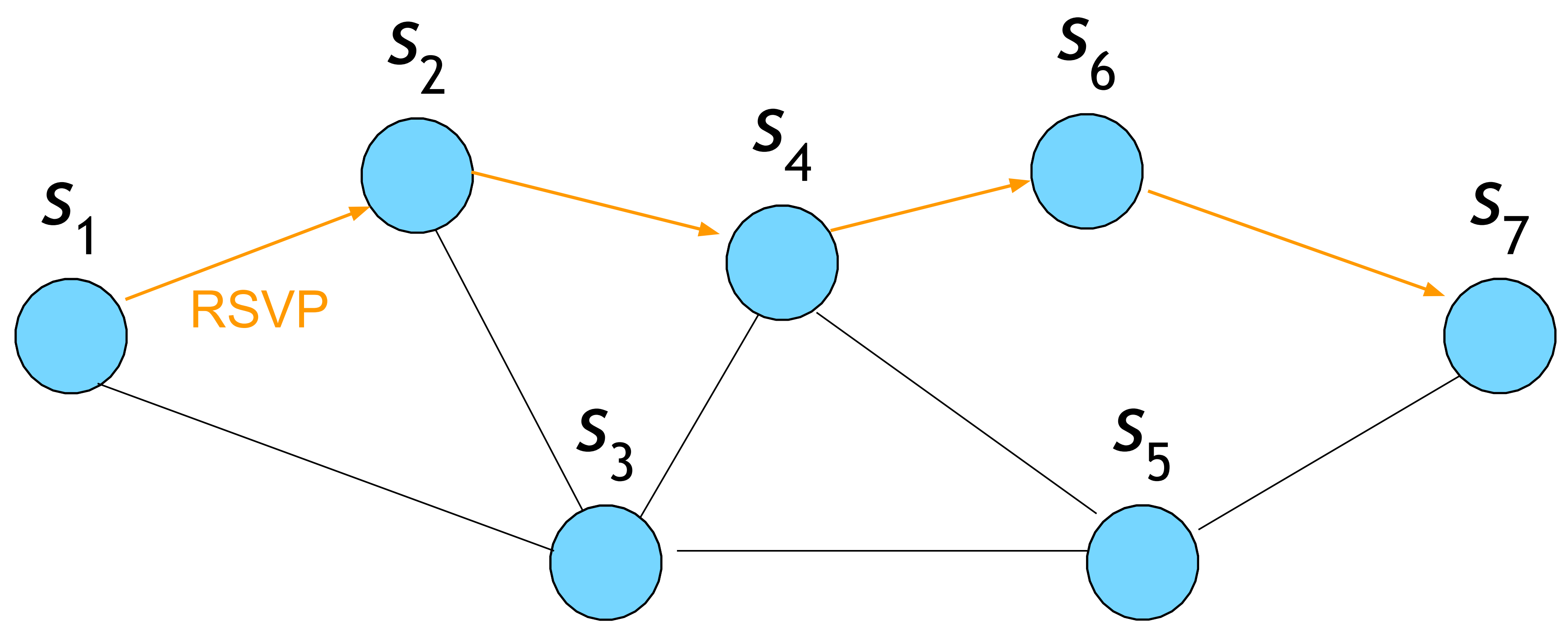
Hardware-Based RSVP

Bandwidth reservations throughout a given path:

- Quality of Service guarantees (IntServ)
- Establishment of virtual circuits (MPLS)

Exclusive data plane implementation:

- Personalized headers
- Header stacks
- Registers
- Bloom filters



Faster and more scalable than traditionally

Quick overview of the proposals

Albert

Thomas

Roland

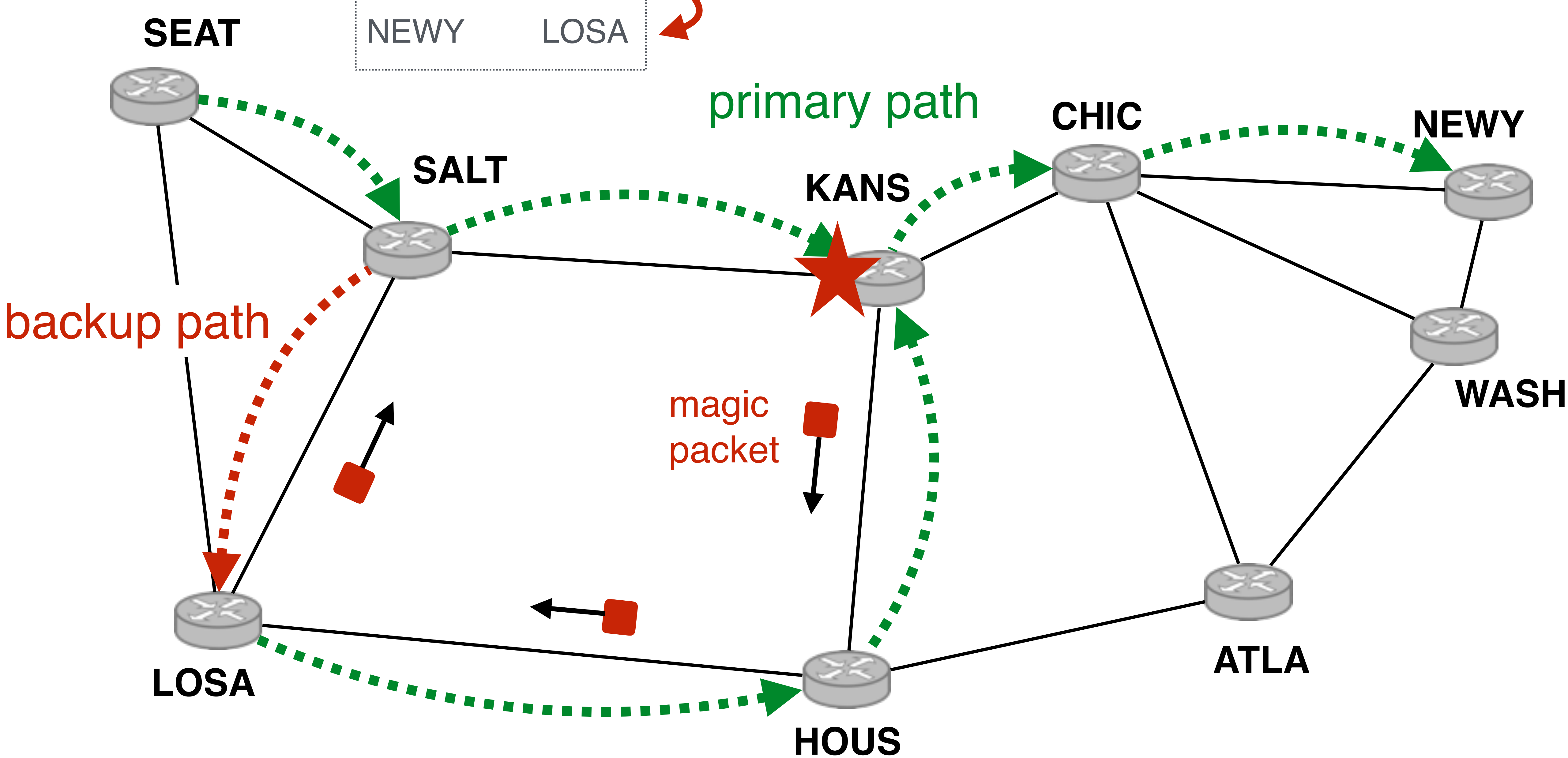
Alexander

Edgar

Proposal #2: Data-plane Driven Network Convergence

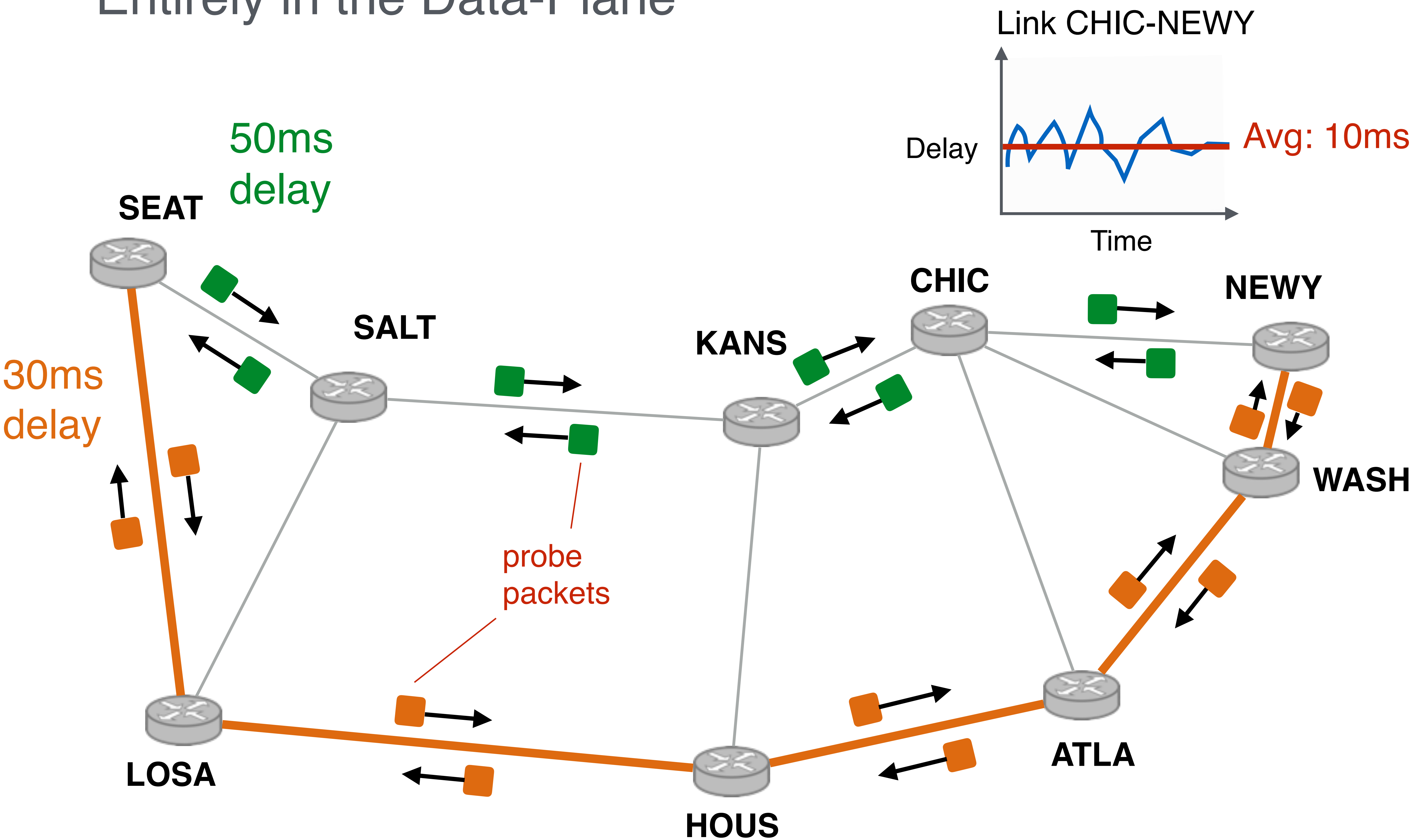
fw table at SALT

dest.	next-hop
~~NEWY~~	~~KANS~~
NEWY	LOSA



Proposal #3: Delay-based Routing

Entirely in the Data-Plane



Quick overview of the proposals

Albert

Thomas

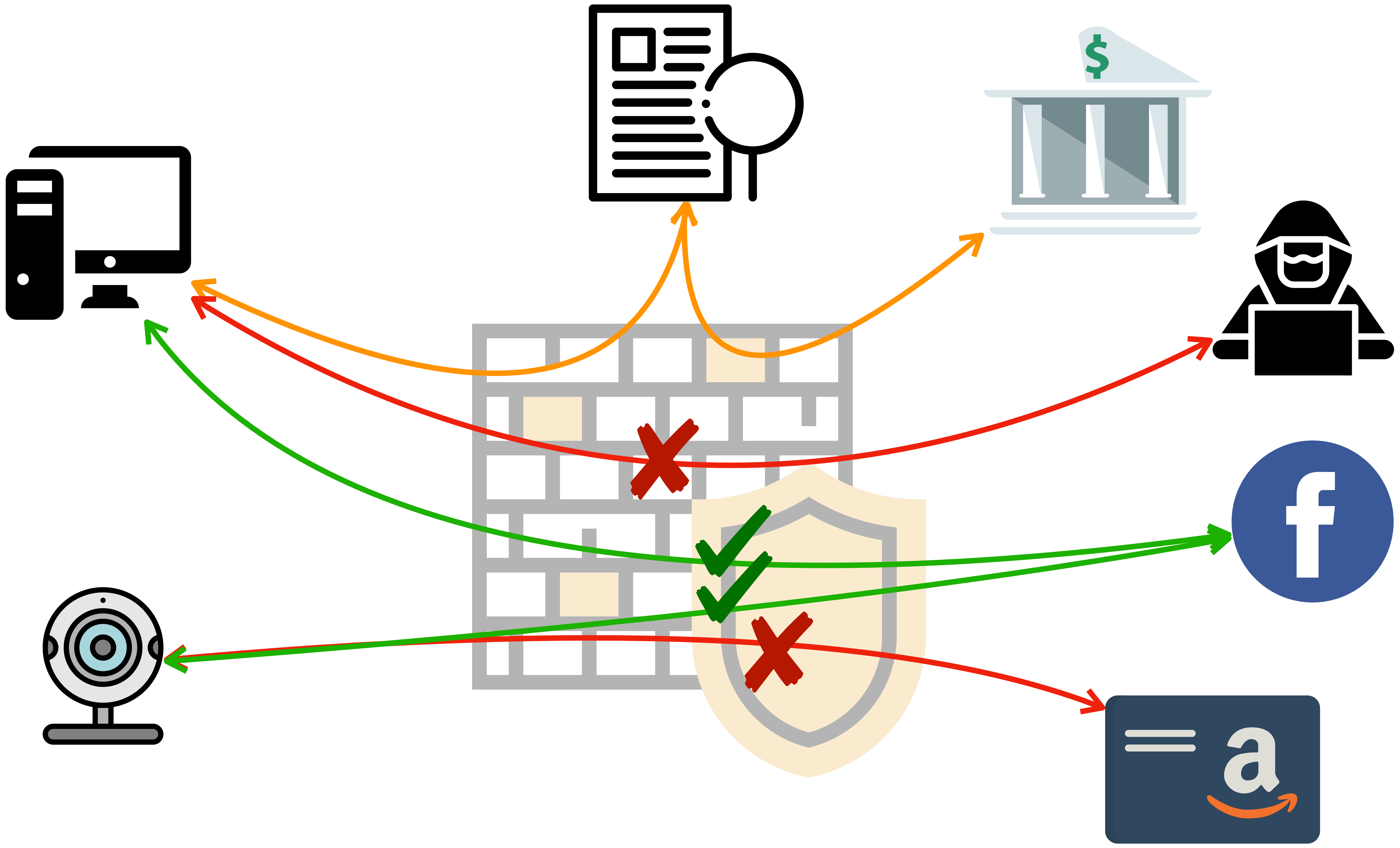
Roland

Alexander

Edgar

Proposal #4

Advanced stateful firewall

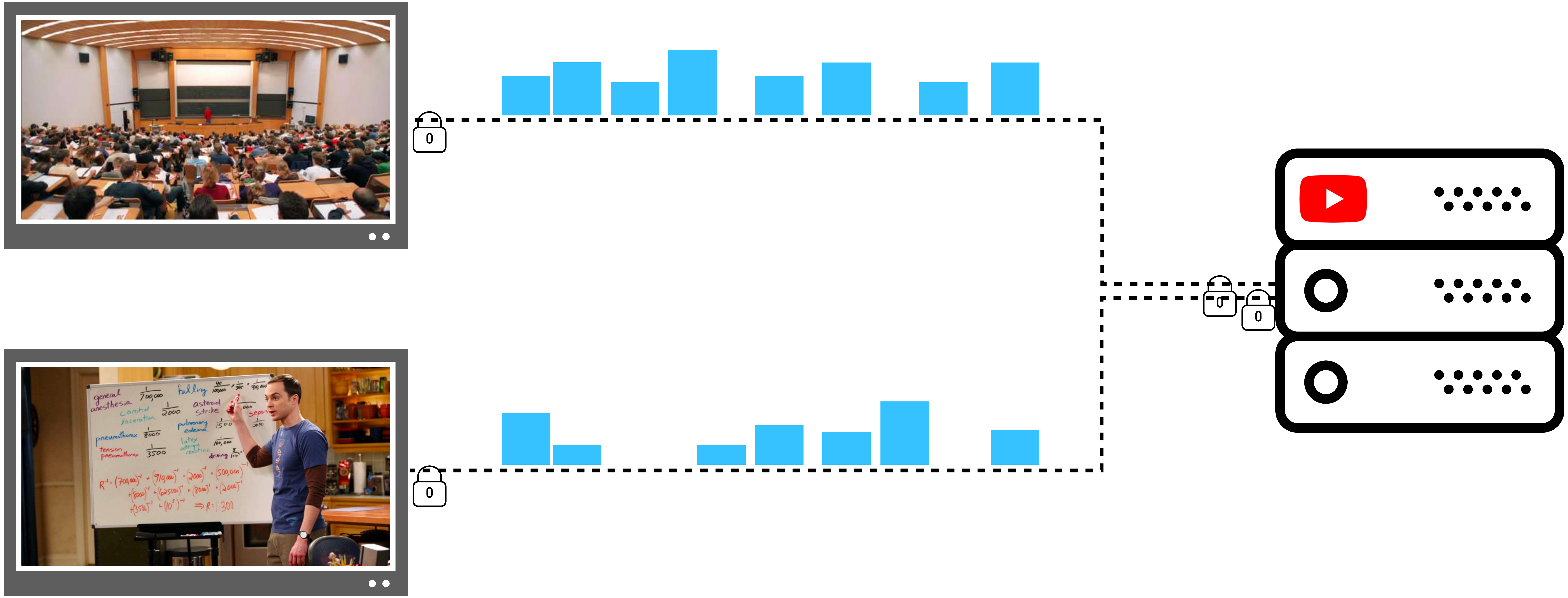


- ✓ Fine-grained access policies
- ✓ Deep packet inspection (DPI)
- ✓ VPN
- ✓ Attack detection
- ✓ Spoofing detection
- ✓ (add your idea here)

…

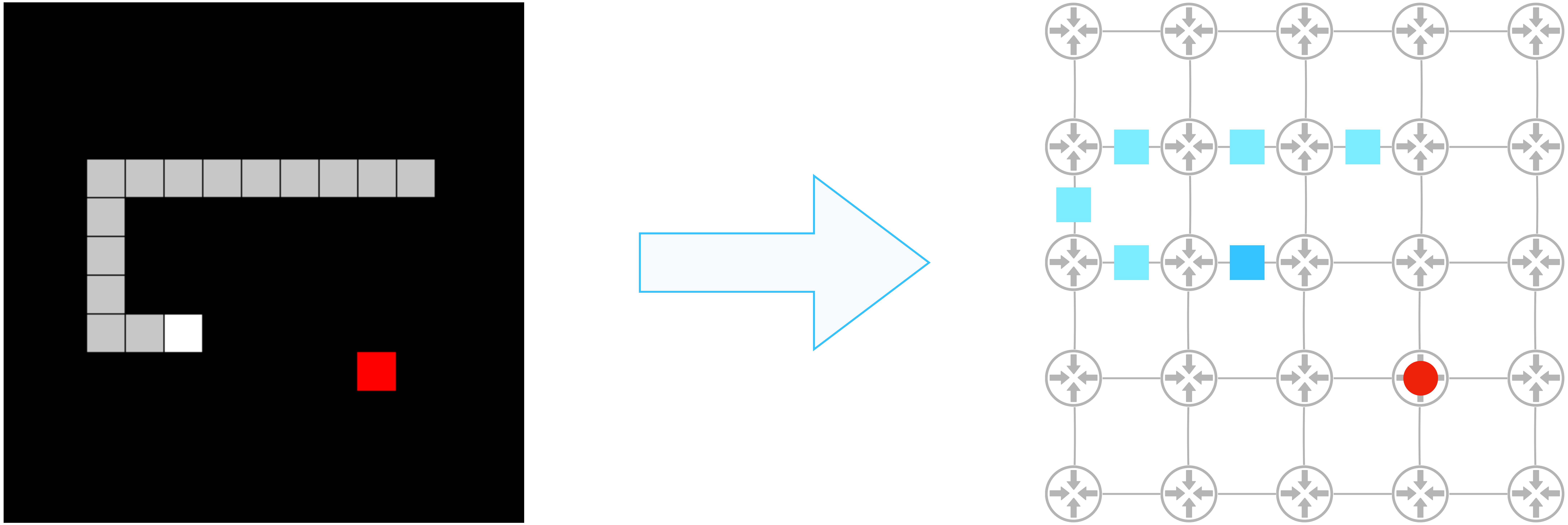
Proposal #5

I know what you're seeing now



Proposal #6

Playing snake in the data plane



Quick overview of the proposals

Albert

Thomas

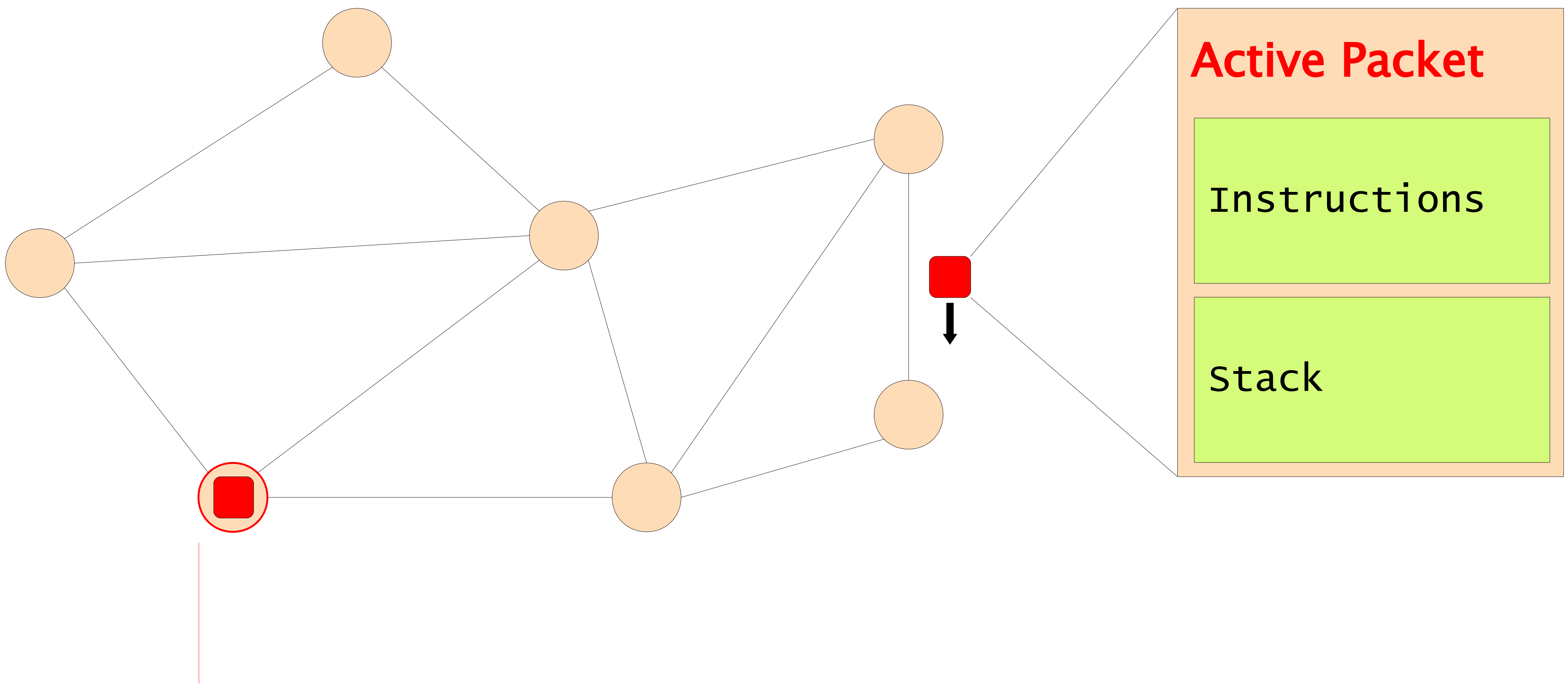
Roland

Alexander

Edgar

Proposal #7

In **Active Networks**, packets carry **programs.**

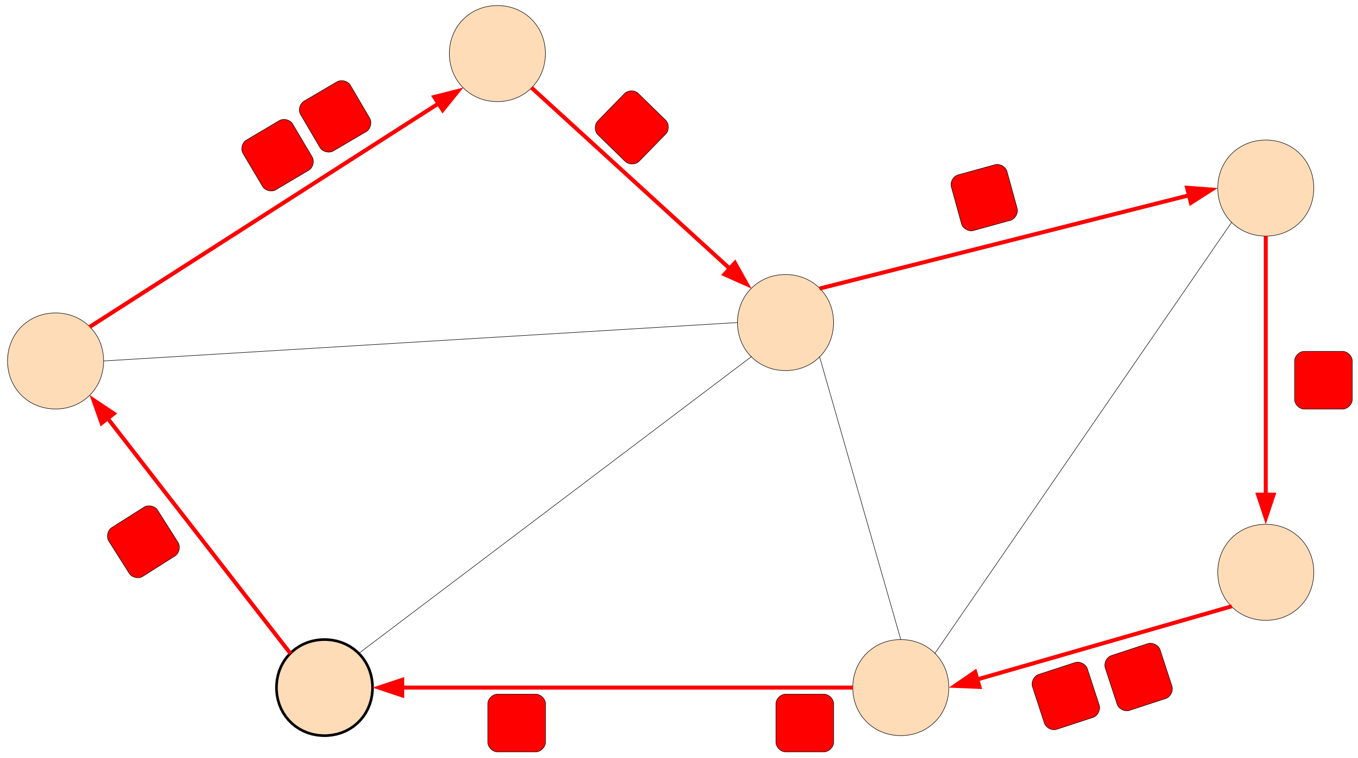


The programs **are executed**

on each switch along the path

Proposal #8

Storing data in the cloud ~~the right way~~!



Store data in a

forwarding loop

Quick overview of the proposals

Albert

Thomas

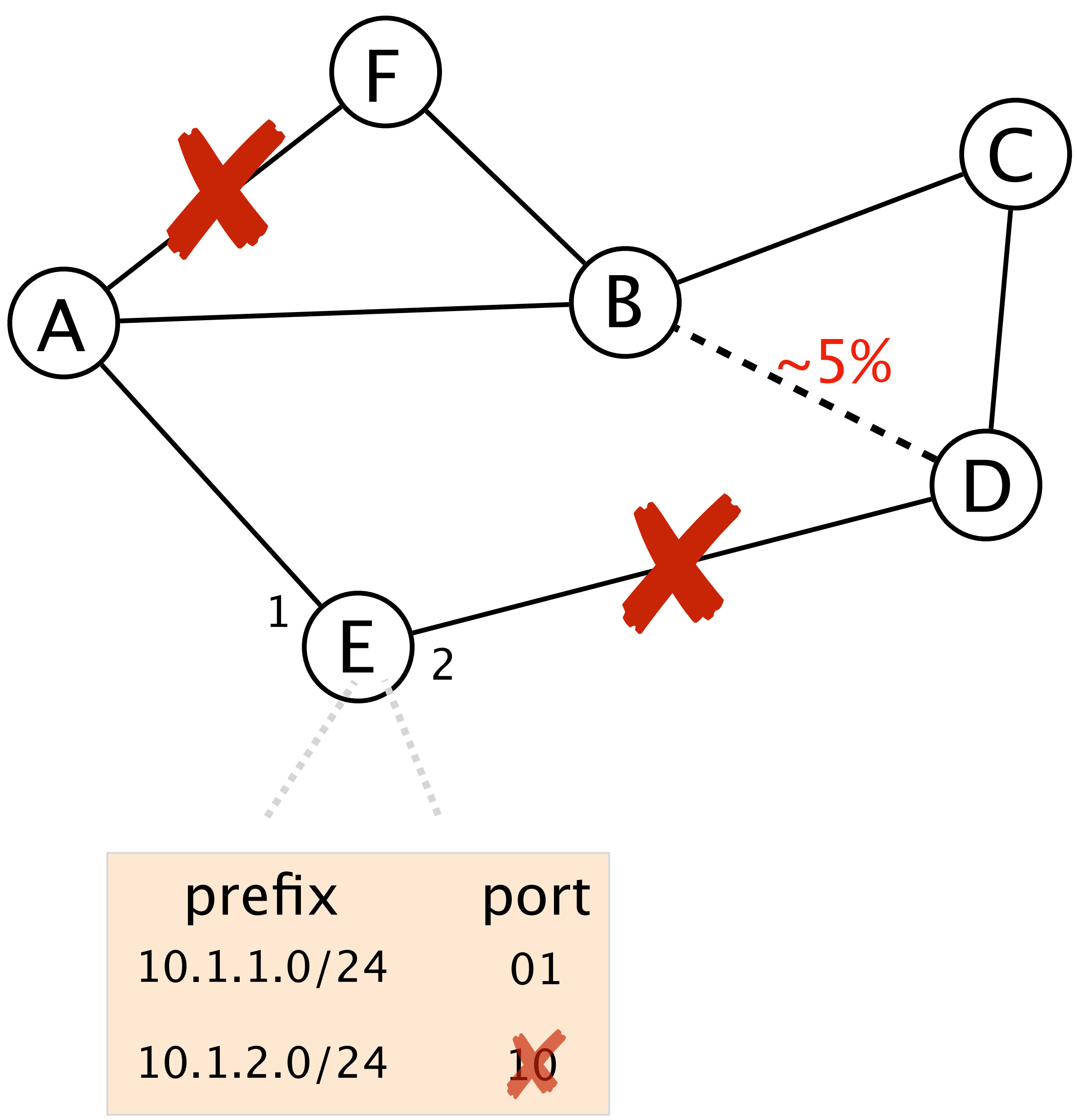
Roland

Alexander

Edgar

Proposal #9

Data Plane Failure Detection



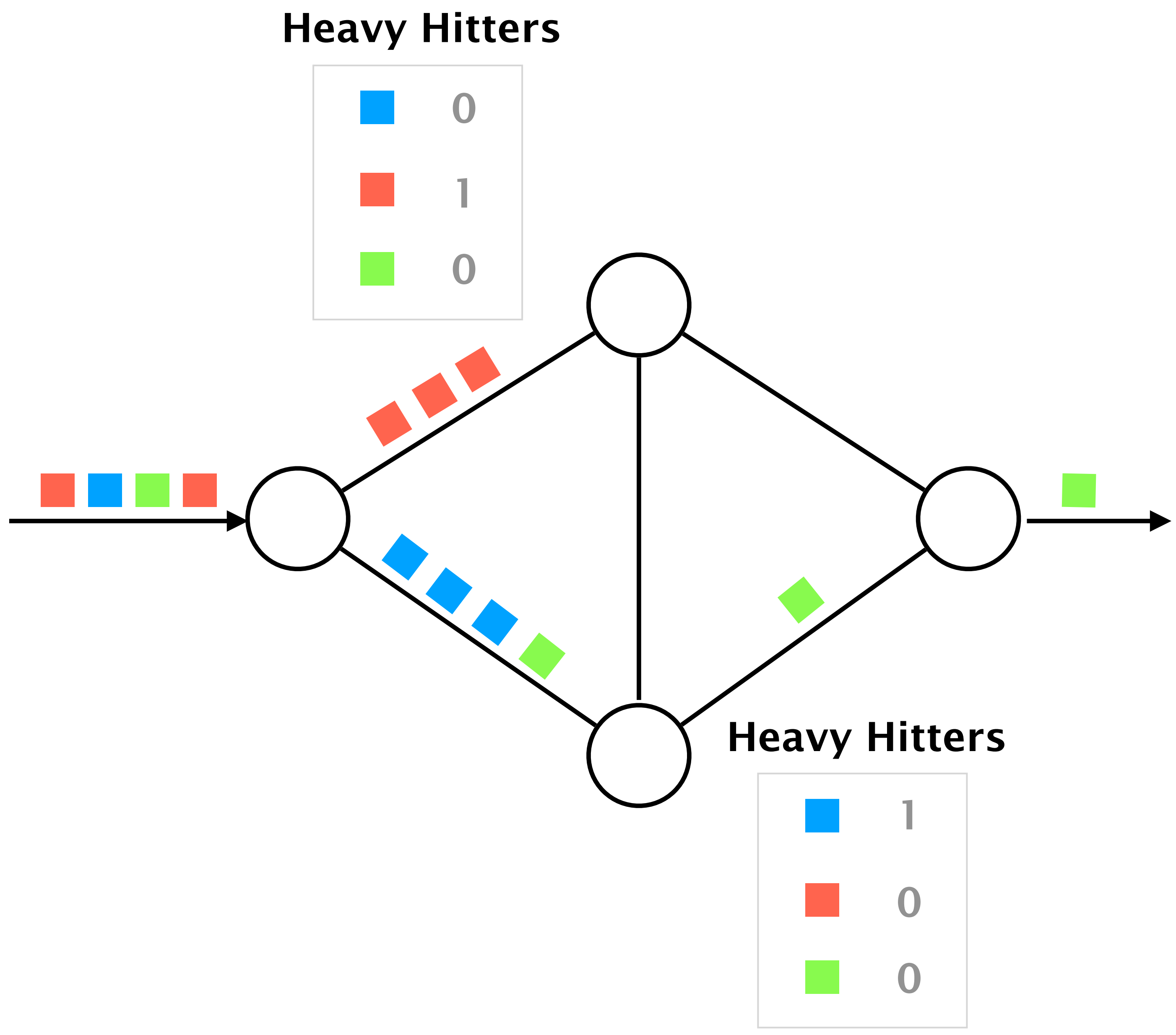
Detect local and remote link failures (A–C)

Detect random packet drops (B–C)

Detect corrupted table entries (E)

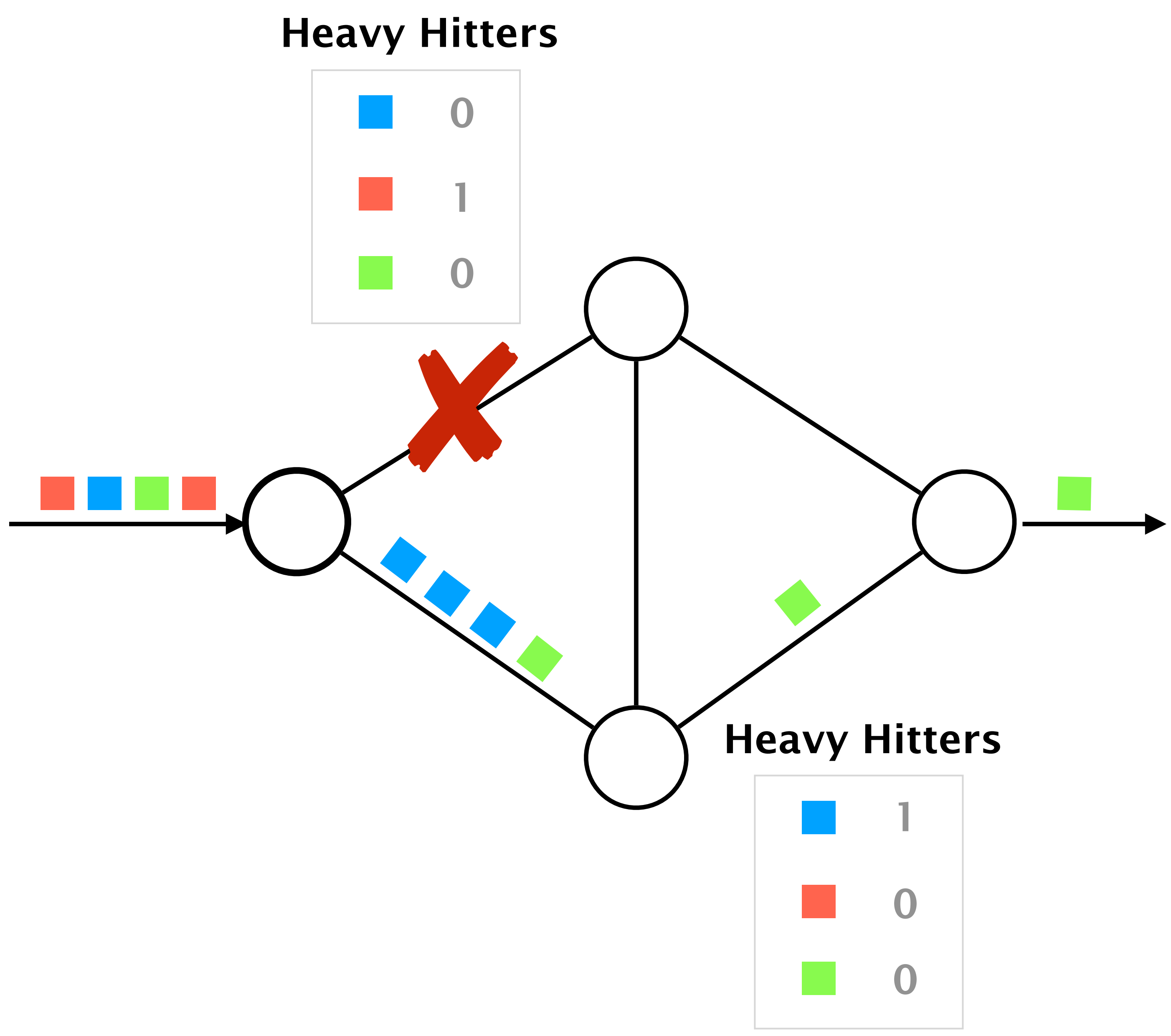
Proposal #10

Stateful Application Migration



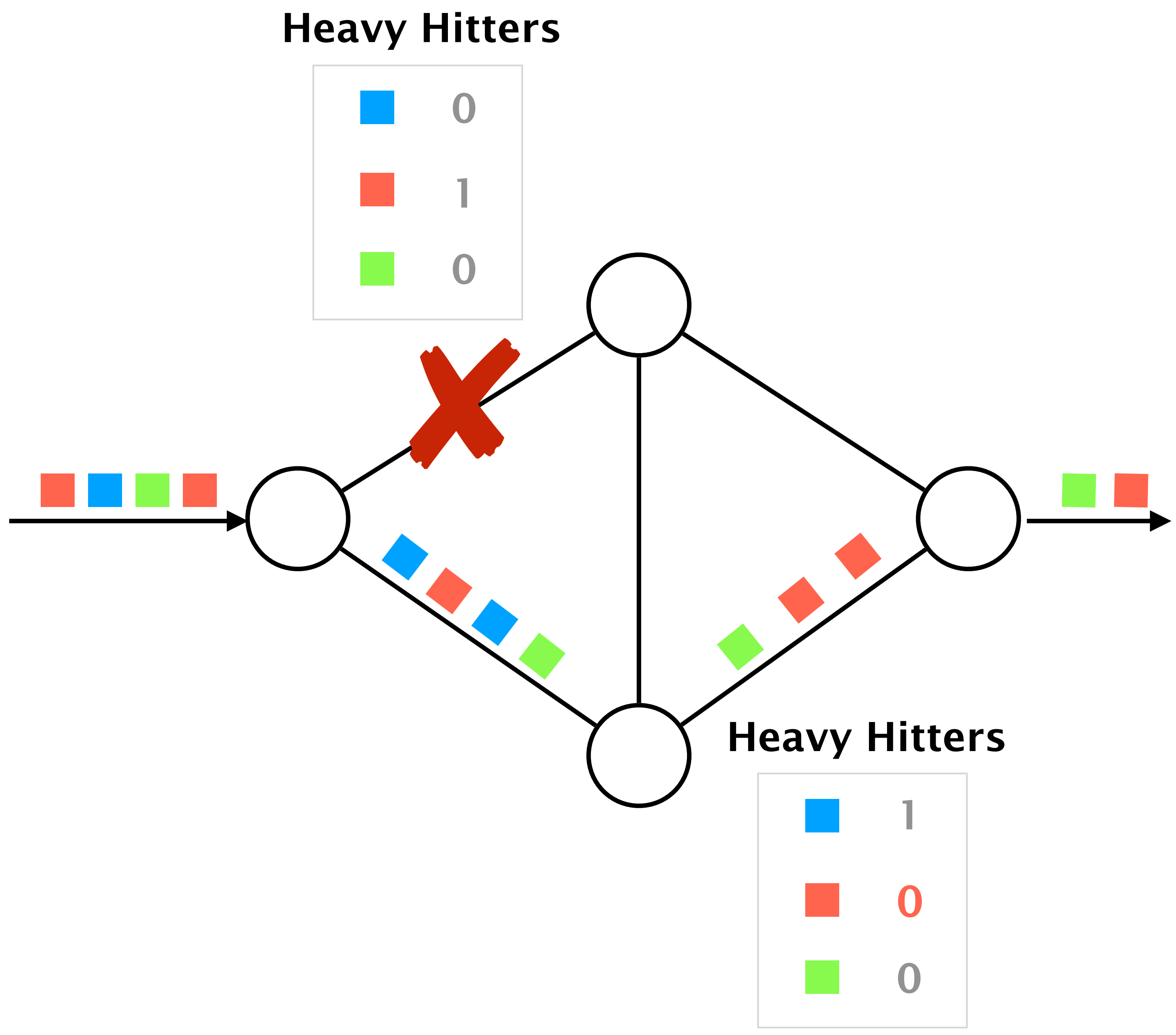
Proposal #10

Stateful Application Migration



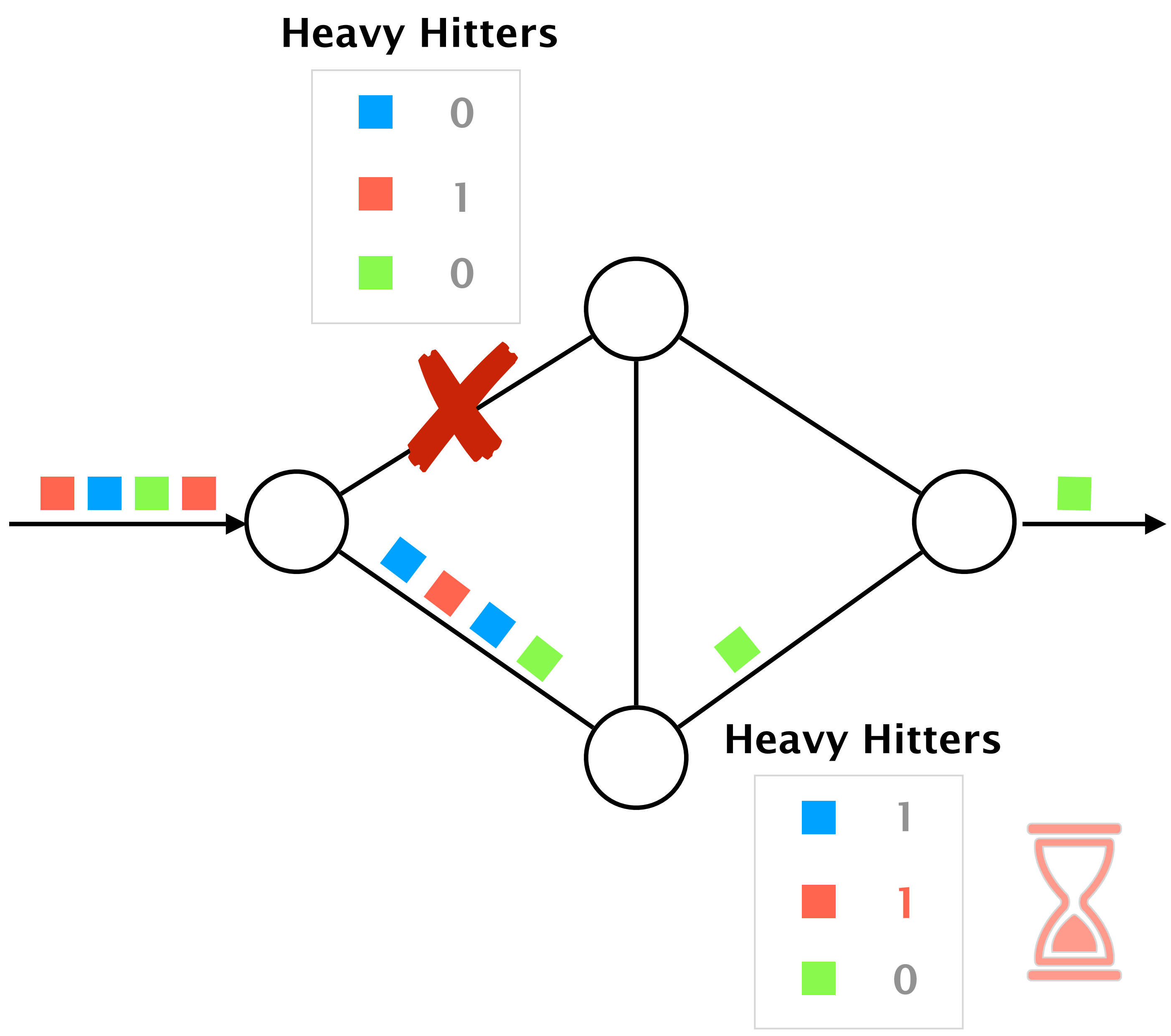
Proposal #10

Stateful Application Migration



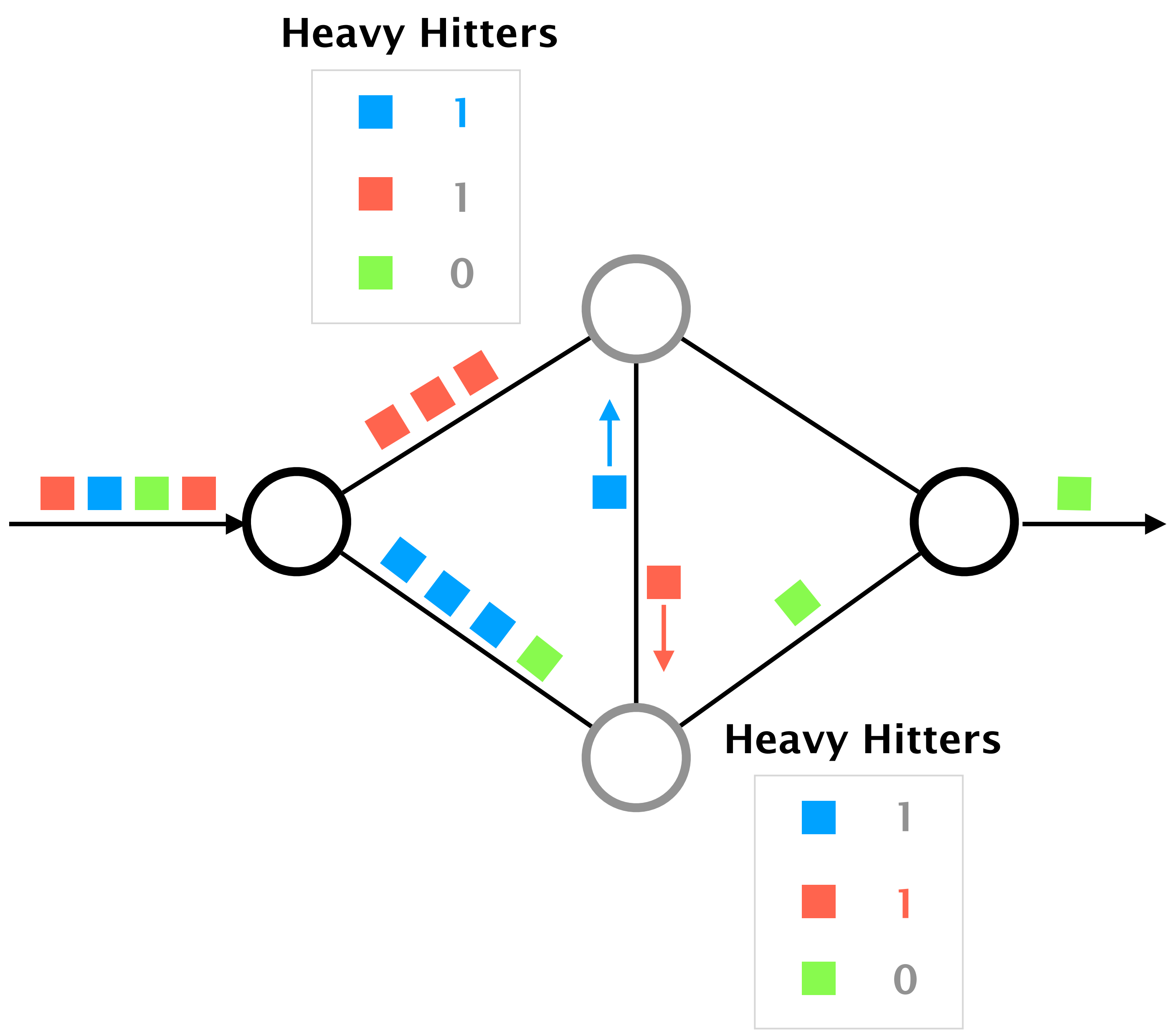
Proposal #10

Stateful Application Migration



Proposal #10

Stateful Application Migration



Proposal #11

P4 Switch

Management and Configuration API

Control Plane

Basic Features	Advanced Features
l2 forwarding, learning, multicast	Spanning Tree Protocol
ipv4, ipv6, l3 multicast	netflow, sFlow or similar
ECMP, Weighted ECMP	VXLAN, MPLS, Gre
ICMP	DHCP Server
ARP	DNS Cache
ECN	Simple Firewall
Simple QoS	NAT

Data Plane

Advanced Topics in Communication Networks

Programming Network Data Planes

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich

Nov 1 2018