Advanced Topics in Communication Networks

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich

Tue 4 Oct 2022

Last week on

Advanced Topics in Communication Networks

We dove into the P4 ecosystem and looked at more advanced language constructs

P4 environment	P4 language	stateful objects
What is needed to program in P4?	Deeper-dive into the language constructs	How do we maintain state in P4?

P4 environment

P4 language

stateful objects

What is needed to program in P4?

$P4_{16}$ introduces the concept of an architecture

P4 Target

hardware or software entity
being programmed

P4 Architecture

a programming model
for a particular target

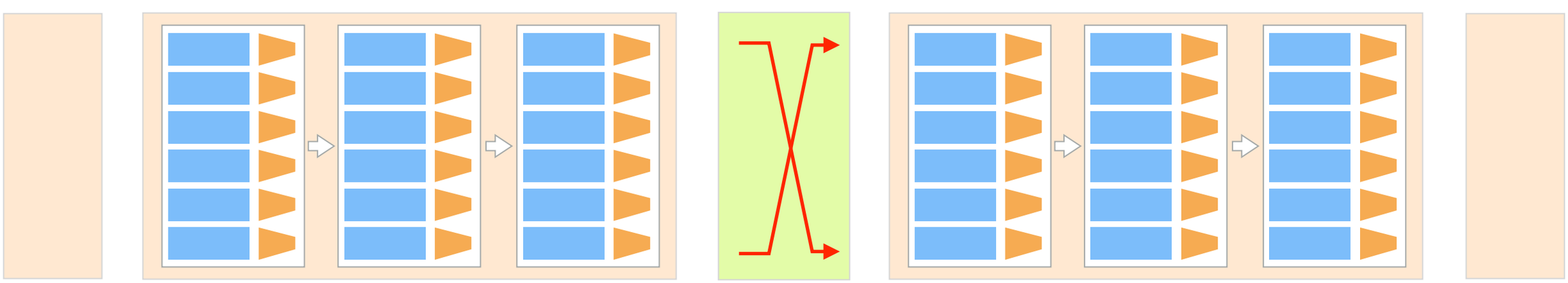
Protocol Independent Switch Architecture (PISA) for high-speed programmable packet forwarding

Ingress

Match-Action Pipeline

Egress

Match-Action Pipeline



Parser

Switching logic

crossbar, shared buffers, …

Deparser

Each architecture defines the metadata it supports, including both standard and intrinsic ones

v1model

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    error parser_error;
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
    bit<32> recirculate_flag;
}
```

more info https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4

Each architecture also defines a list of "externs",
i.e. blackbox functions whose interface is known

Most targets contain specialized components

which cannot be expressed in P4 (e.g. complex computations)

At the same time, $P4_{16}$ should be target-independent

In $P4_{14}$ almost 1/3 of the constructs were target-dependent

Think of externs as Java interfaces

only the signature is known, not the implementation

P4 environment

P4 language

stateful objects

Deeper dive into
the language constructs (*)

(*) full info https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

P4$_{16}$ is a statically-typed language with

base types and operators to derive composed ones

`bool`	Boolean value
`bit<W>`	Bit-string of width W
`int<W>`	Signed integer of width W
`varbit<W>`	Bit-string of dynamic length ≤W
`match_kind`	describes ways to match table keys
`error`	used to signal errors
`void`	no values, used in few restricted circumstances
`float`	not supported
`string`	not supported

P4$_{16}$ is a statically-typed language with

base types and operators to derive composed ones

P4 operations are similar to C operations and vary depending on the types (unsigned/signed ints, …)

- arithmetic operations `+, -, *`
- logical operations `~, &, |, ^, >>, <<`
- non-standard operations `[m:l]` Bit-slicing

  `++` Bit concatenation

✘ *no* division and modulo (can be approximated)

more info https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

Variables have local scope and their values is not maintained across subsequent invocations

important

variables *cannot* be used to maintain state between different network packets

instead

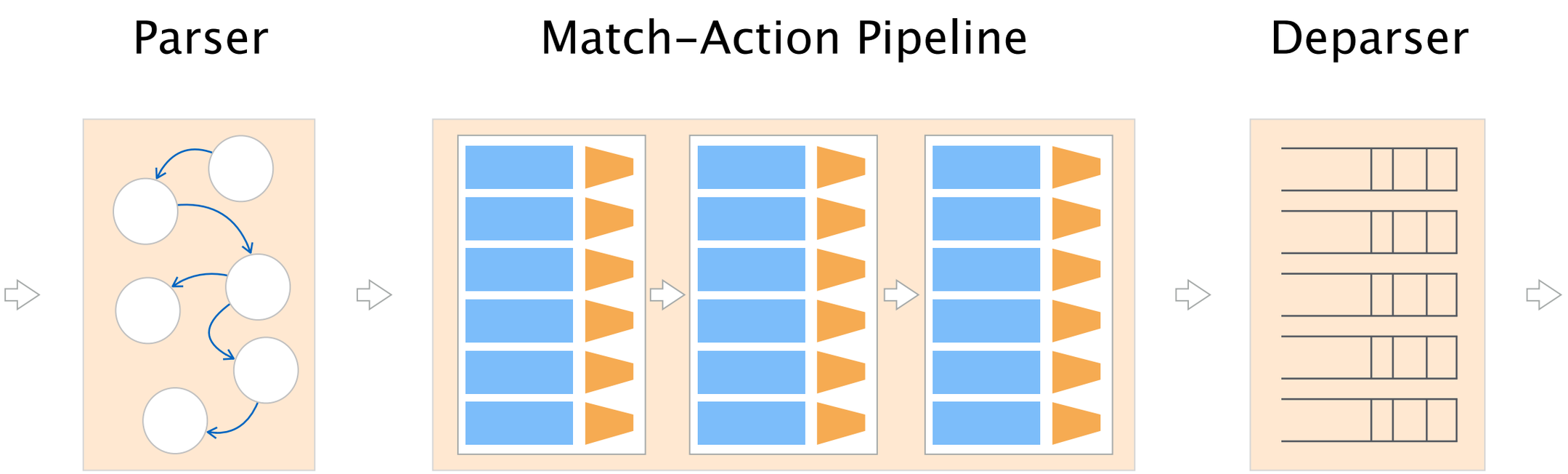
to maintain state

you can only use two stateful constructs

- tables — modified by control plane
- extern objects — modified by control plane & data plane

more on this soon

We then started to look at more advanced constructs pertaining to each of the three building blocks



This week on

Advanced Topics in Communication Networks

We'll finish our exploration of the P4 language then look at our first technique

stateful objects

How do we maintain state in P4?

load balancing

How do we balance traffic in a network?

stateful objects

load balancing

How do we maintain state in P4?

(last week's slides)

How do we balance traffic in a network?

stateful objects

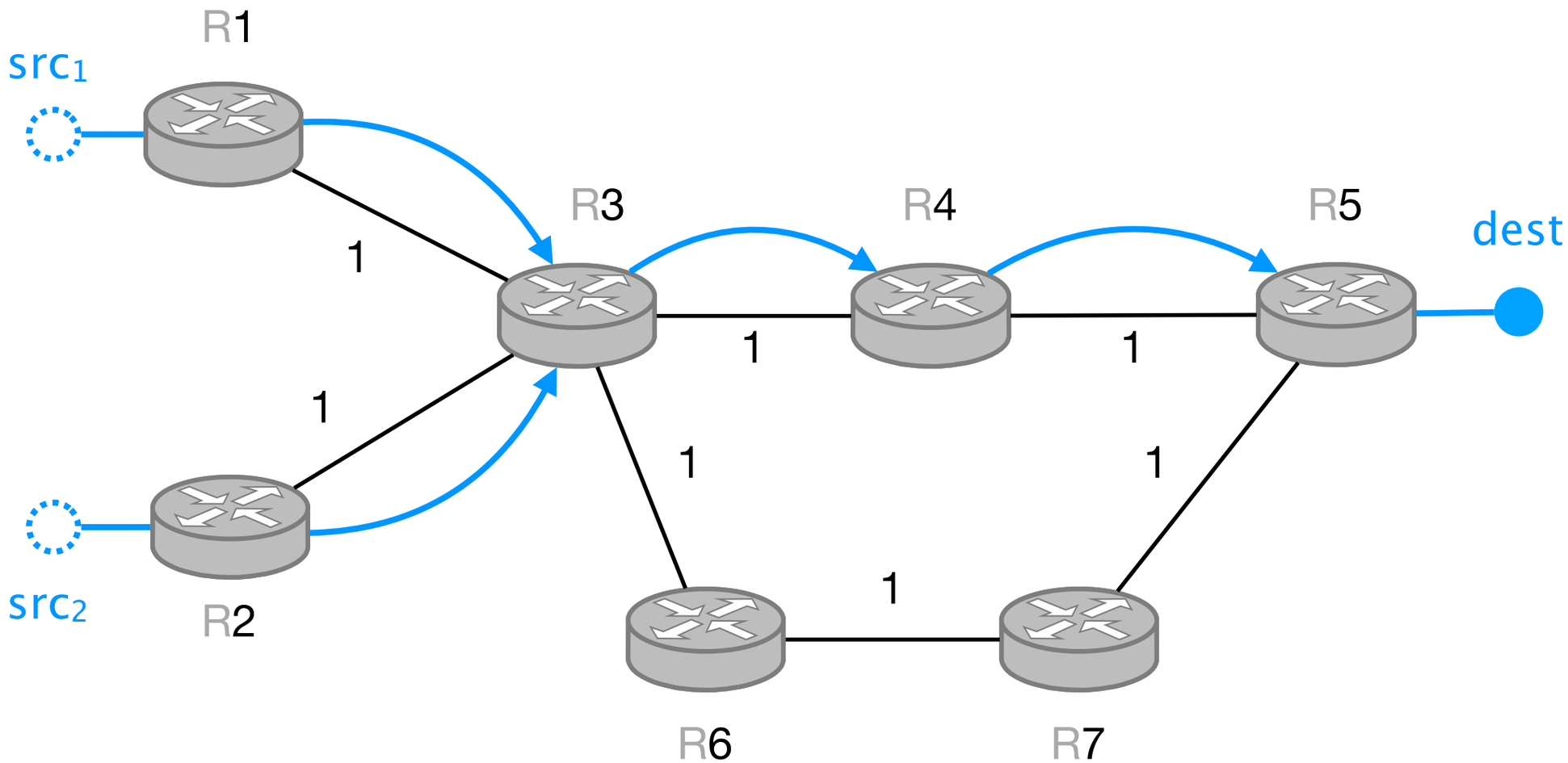
load balancing

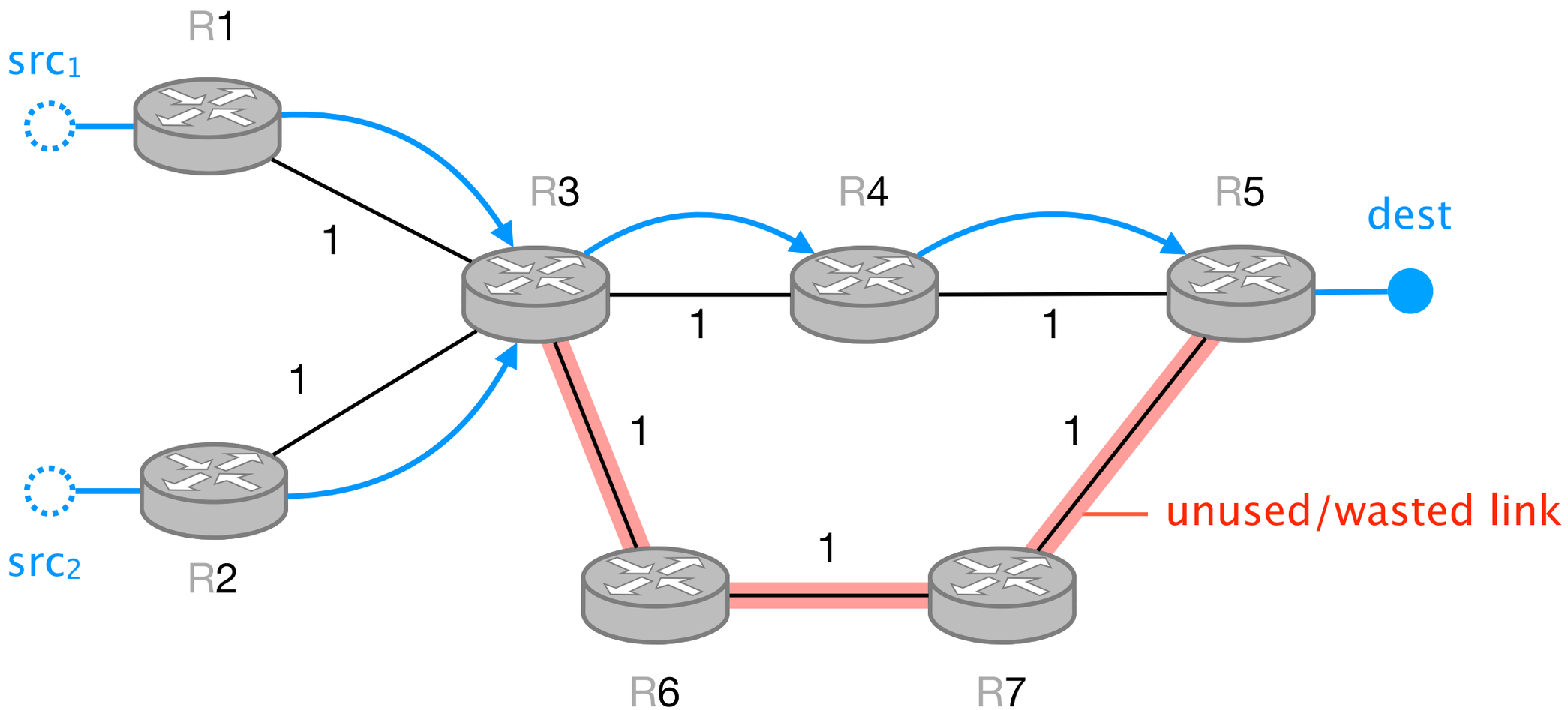
How do we maintain state in P4?

How do we balance traffic in a network?

Shortest path routing does not always lead to good network utilization

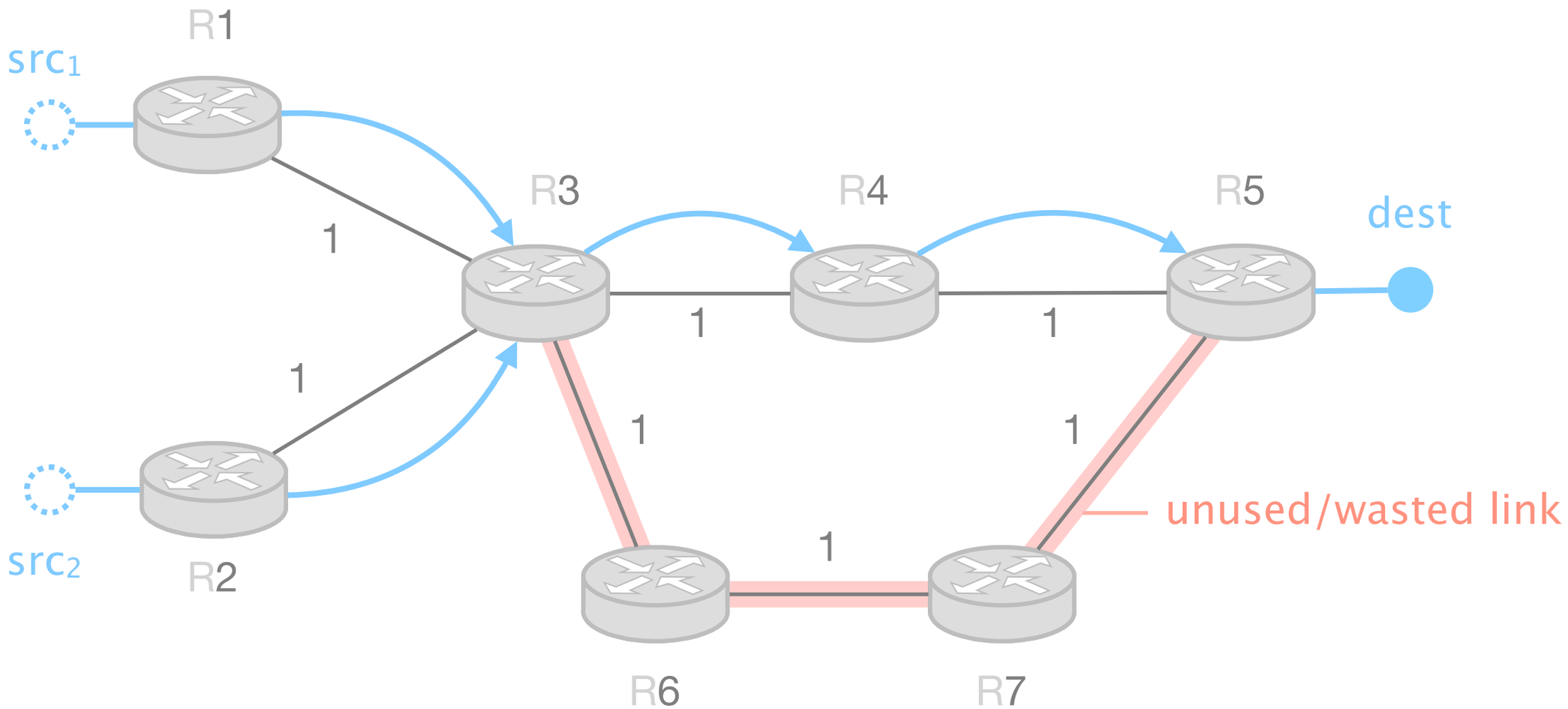
Shortest path routing does not always lead to good network utilization




R1
src1
R3
R4
R5
dest
1
1
1
1
1
1
1
1
unused/wasted link
src2
R2
R6
R7

How can we optimize network utilization?

How can we react to changes in traffic conditions?



Traffic engineering encompasses a set of techniques to control the flow of traffic in data networks

control…

… at the node-level	allow a node to use >1 paths to reach a destination
… at the network-level	route specific traffic flows onto specific paths inside the network

control…

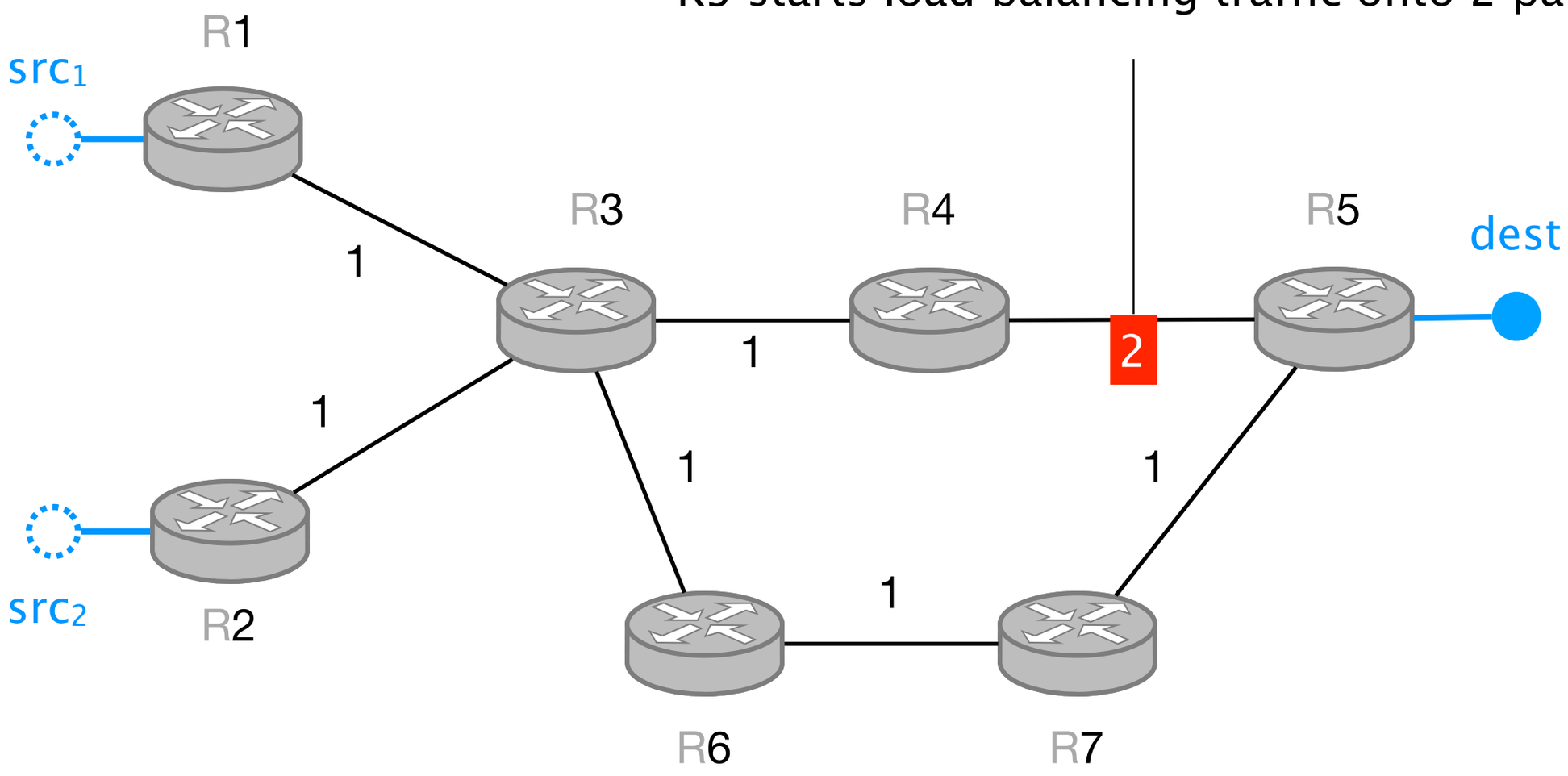
… at the node-level	allow a node to use >1 paths to reach a destination
… at the network-level	route specific traffic flows onto specific paths inside the network

Equal-Cost Multi-Path (ECMP) is a load-balancing strategy in which nodes split traffic over all best paths

Equal-Cost Multi-Path (ECMP) is a load-balancing strategy in which nodes split traffic over all best paths

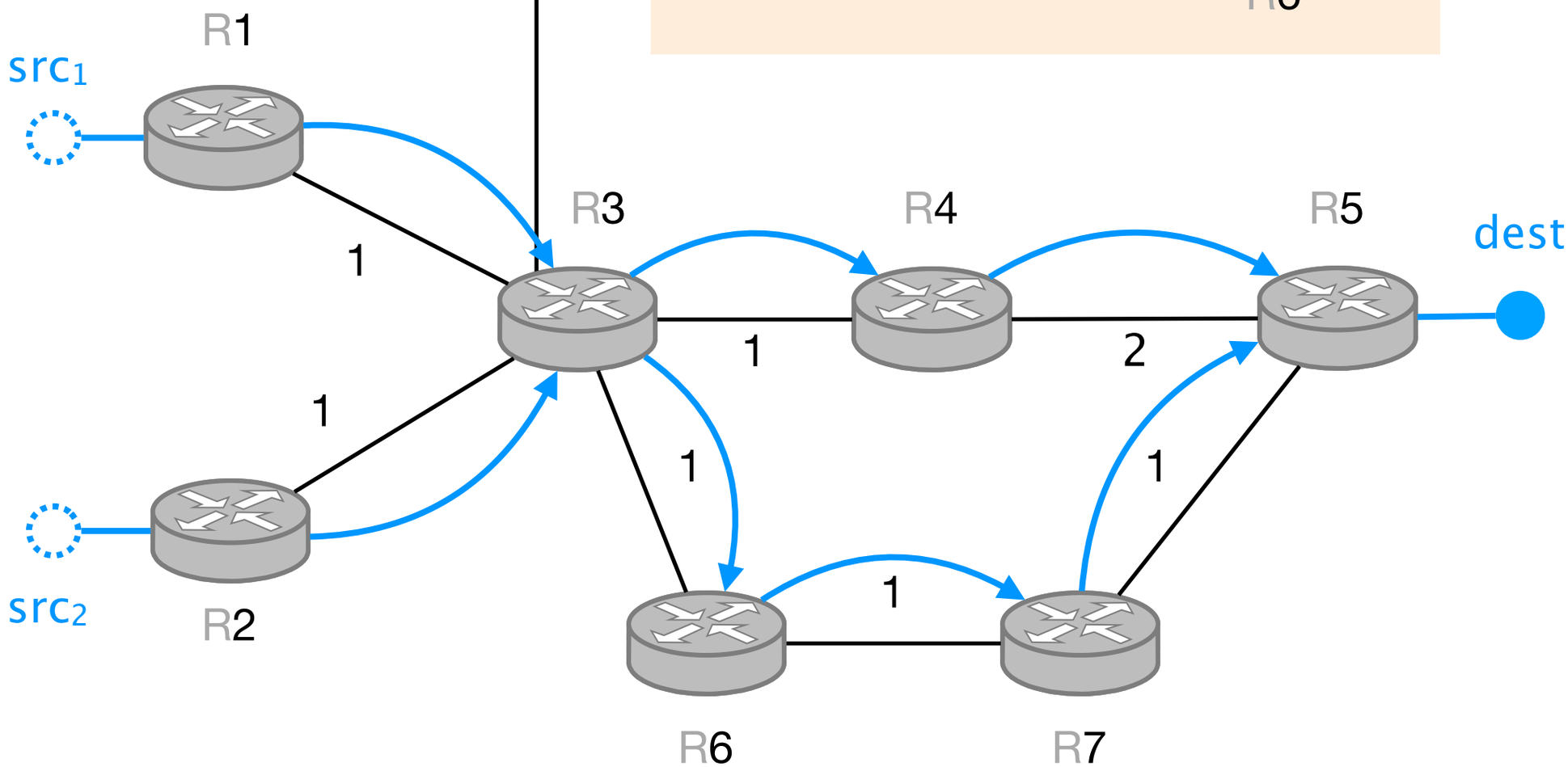
in OSPF, best == shortest

By adapting this weight to 2,
R3 starts load balancing traffic onto 2 paths



R3's routing table

pfx	min_cost	next_hop
dest	3	R4
		R6



Equal-Cost Multi-Path (ECMP) is a load-balancing strategy in which nodes split traffic over all best paths

how?

ECMP relies on *stateless* hash functions to split traffic

ECMP next_hop = hash [src_ip, dst_ip, src_port, dst_port, proto] % # next_hops

Property

All packets belonging to the same TCP flows go over the same path

(doing so avoids reordering)

ECMP suffers from two main problems:

hash collisions and asymmetry

ECMP suffers from two main problems:

hash collisions and asymmetry

can cause significant imbalance

if there are few large flows

Let's look at an example in which

ECMP underperforms because of hash collisions

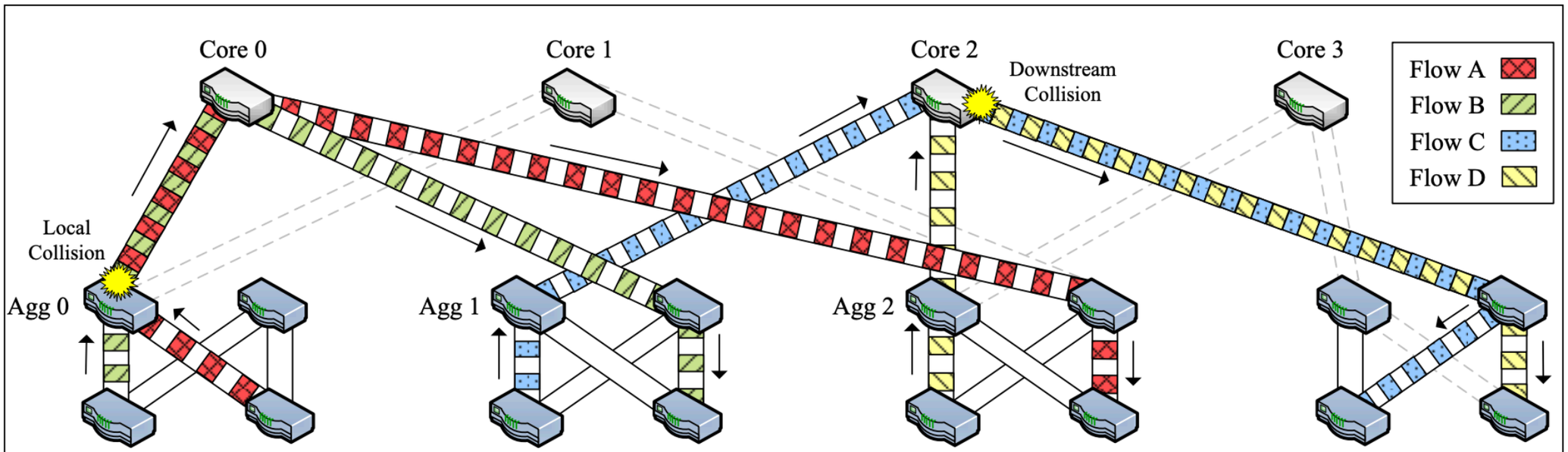
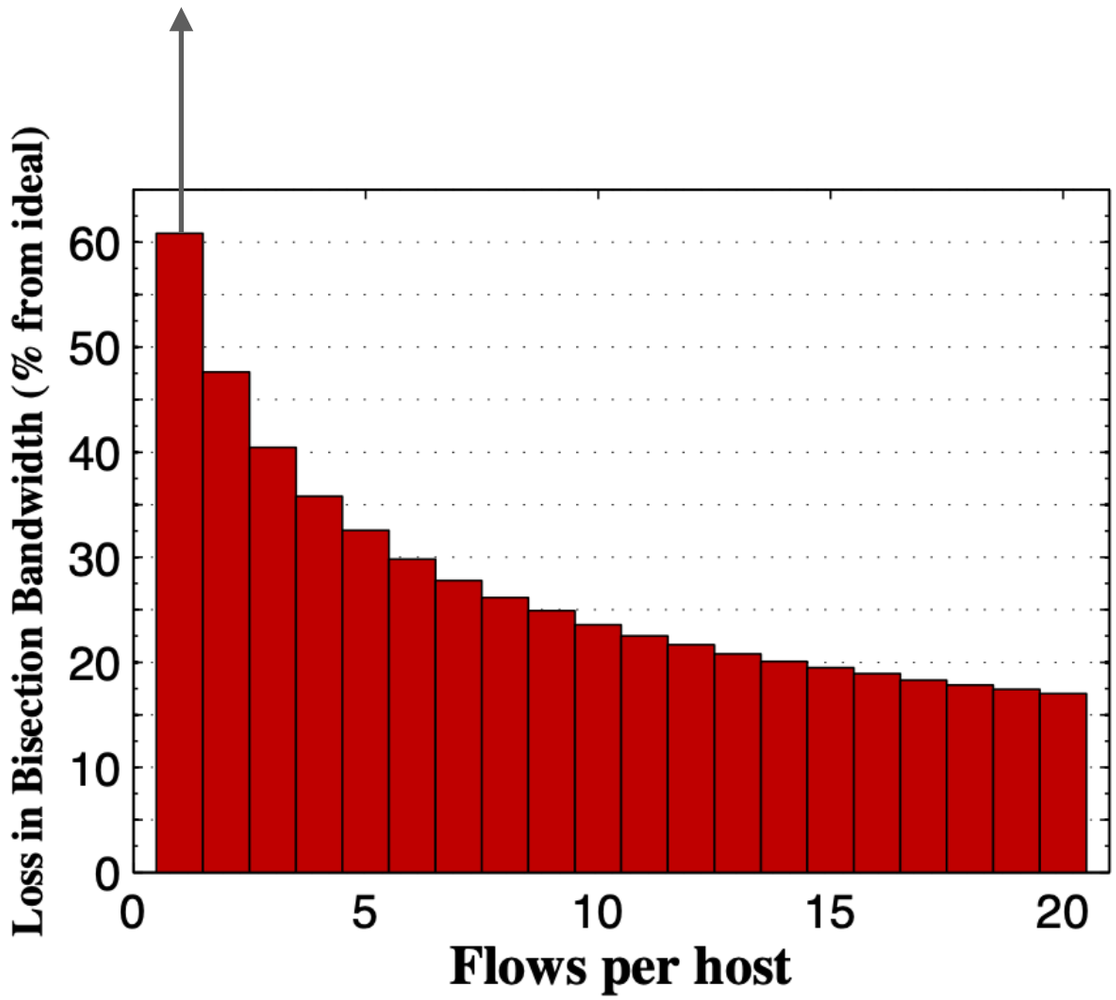


Figure 2: Examples of ECMP collisions resulting in reduced bisection bandwidth. Unused links omitted for clarity.

Hedera: Dynamic Flow Scheduling for Data Center Networks, USENIX NSDI 2010

If each host transfers an equal amount of data to all remote hosts one at a time,
hash collisions reduce the network's bisection bandwidth by an average of 60.8%



Hedera: Dynamic Flow Scheduling for Data Center Networks, USENIX NSDI 2010

across 1000 simultaneous flows

If each host transfers an equal amount of data to all remote hosts ~~one at a time~~,
hash collisions reduce the network's bisection bandwidth by an average of ~~60.8%~~
only 2.5%

across 1000 simultaneous flows

If each host transfers an equal amount of data to all remote hosts ~~one at a time~~,
hash collisions reduce the network's bisection bandwidth by an average of ~~60.8%~~
only 2.5%

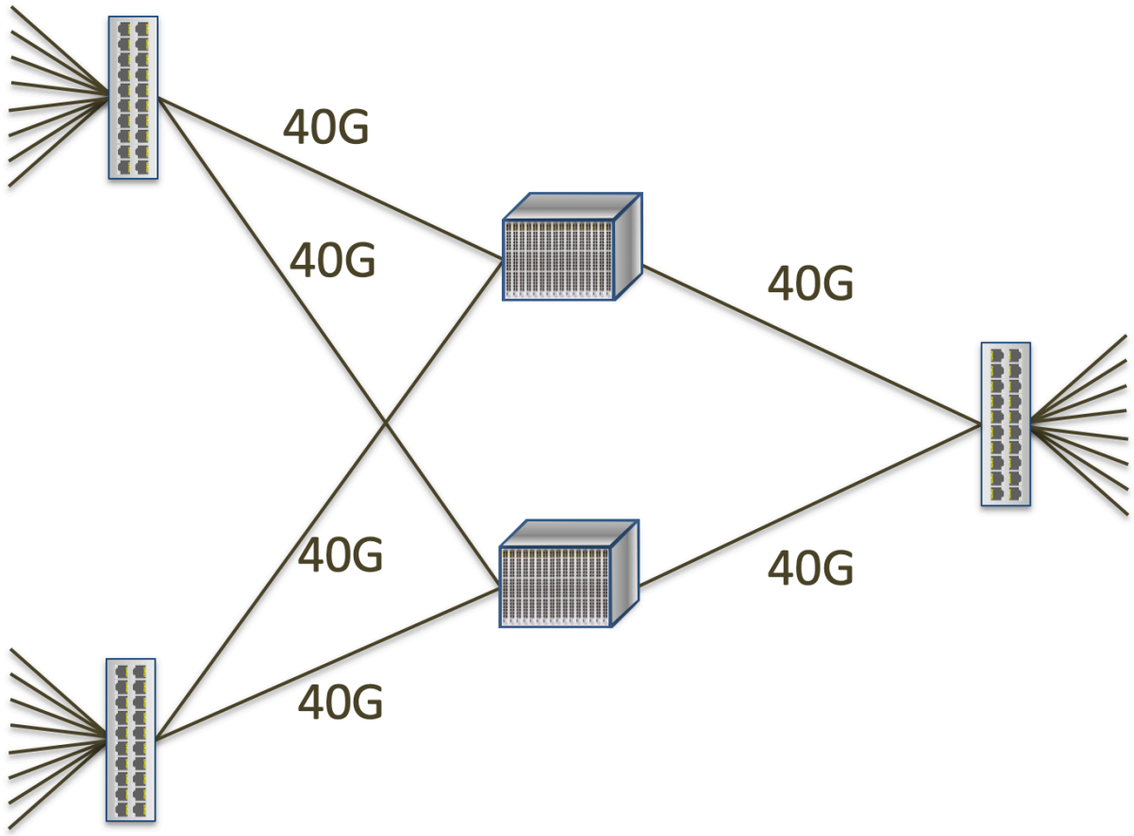
Intuition The cost of a collision decreases with the number of flows

ECMP suffers from two main problems:

hash collisions and asymmetry

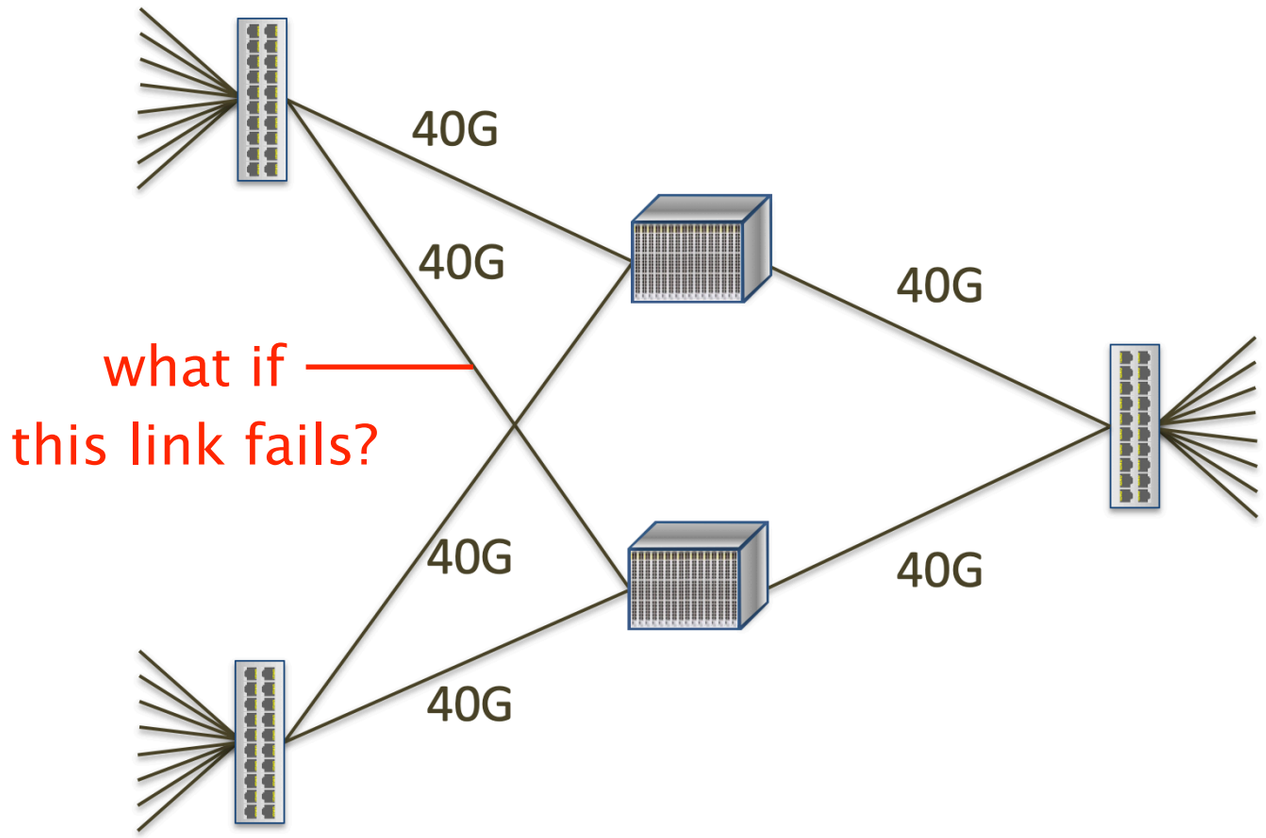
what if the paths we're load balancing onto are not equivalent / symmetric?

Dealing with Asymmetry



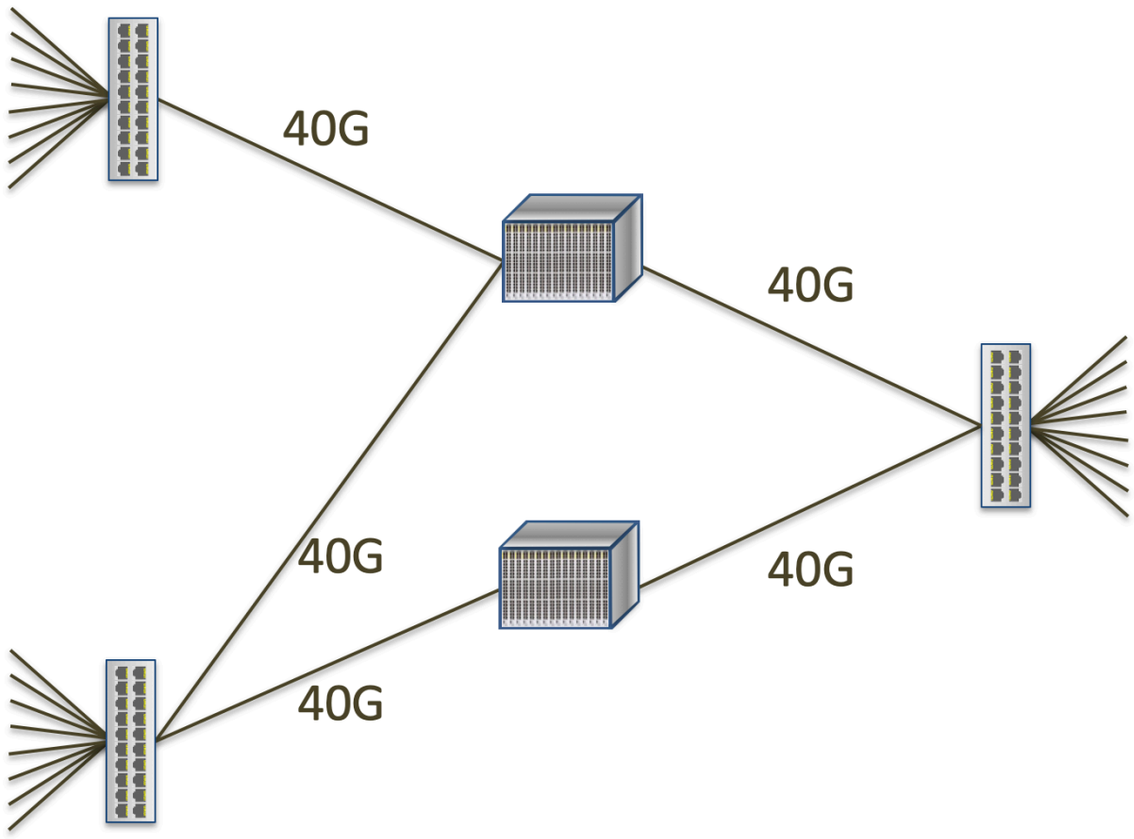
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Dealing with Asymmetry



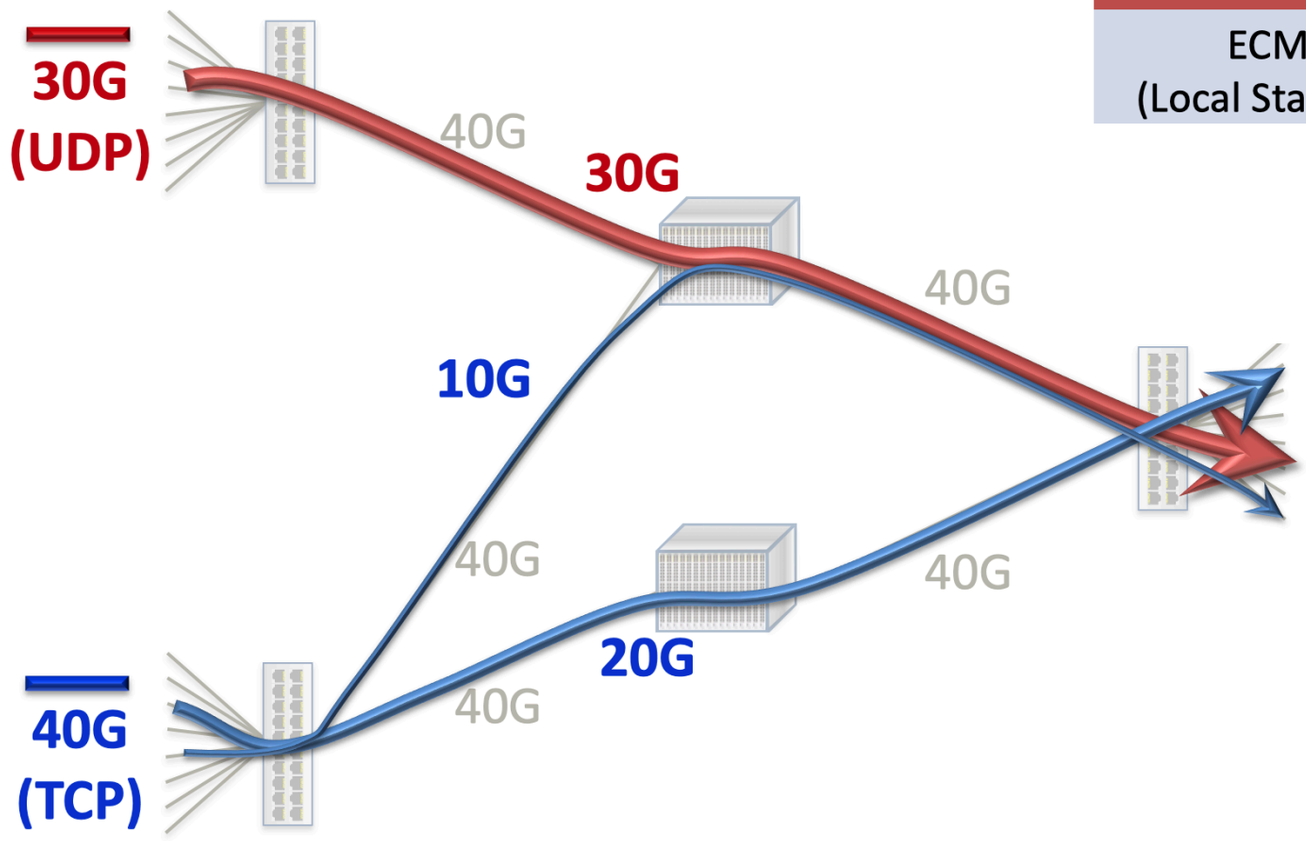
Source: CONGA: Distributed Congestion-Aware Load Balancing for Datacenters, Mohammad Alizadeh et al., 2014

Dealing with Asymmetry



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

Dealing with Asymmetry: ECMP



Scheme	Thrput
ECMP (Local Stateless)	**60G**

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

ECMP load-balancing decisions are suboptiomal because they are too coarse-grained and congestion-agnostic

load-balancing decision…

load-balancing decision…

granularity

congestion-
awareness

load-balancing decision…

granularity

flow

flowlet

packet

congestion-awareness

load-balancing decision…

granularity

flow

flowlet

packet

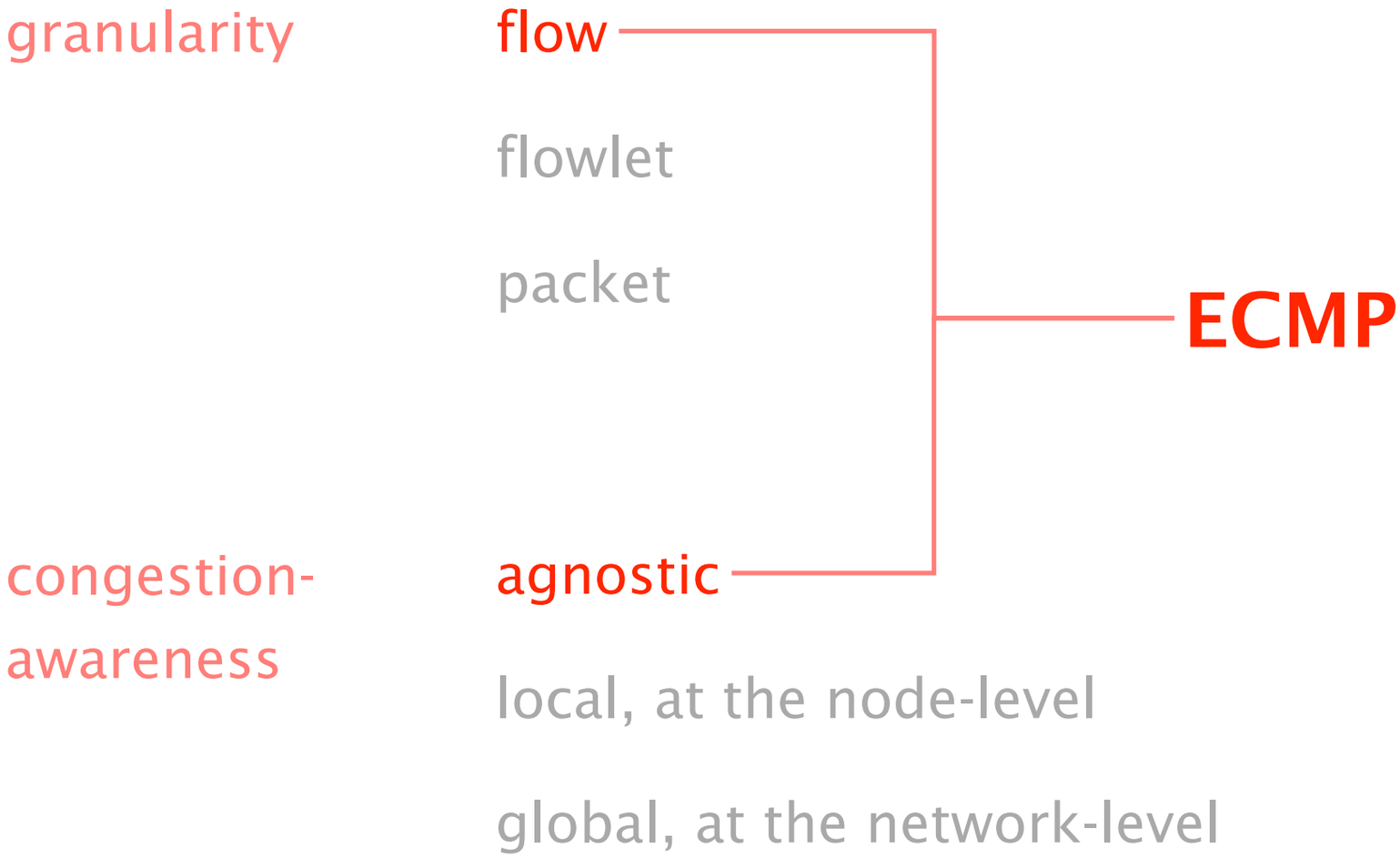
congestion-awareness

agnostic

local, at the node-level

global, at the network-level

ECMP load-balancing decisions are suboptiomal because they are too coarse-grained and congestion-agnostic



load-balancing decision…

granularity

~~flow~~

flowlet

packet

congestion-awareness

~~agnostic~~

local, at the node-level

global, at the network-level

load-balancing decision…

granularity

~~flow~~

flowlet

packet —— finest granularity… but *bad* for TCP because of reordering

congestion-awareness

~~agnostic~~

local, at the node-level

global, at the network-level

load-balancing decision…

granularity

~~flow~~

flowlet

~~packet~~

congestion-awareness

~~agnostic~~

local, at the node-level

global, at the network-level

Flowlets: 

"Flowlets are bursts from same flow separated by at least Δ"

"the main origin of flowlets is the burstiness of TCP at RTT and sub-RTT scales."

Kandula et al, "Dynamic load balancing without packet reordering", (2007)

Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

load-balancing decision…

granularity

~~flow~~

flowlet **best pick!**

~~packet~~

congestion-awareness

~~agnostic~~

local, at the node-level

global, at the network-level

load-balancing decision…

granularity

~~flow~~

flowlet

~~packet~~

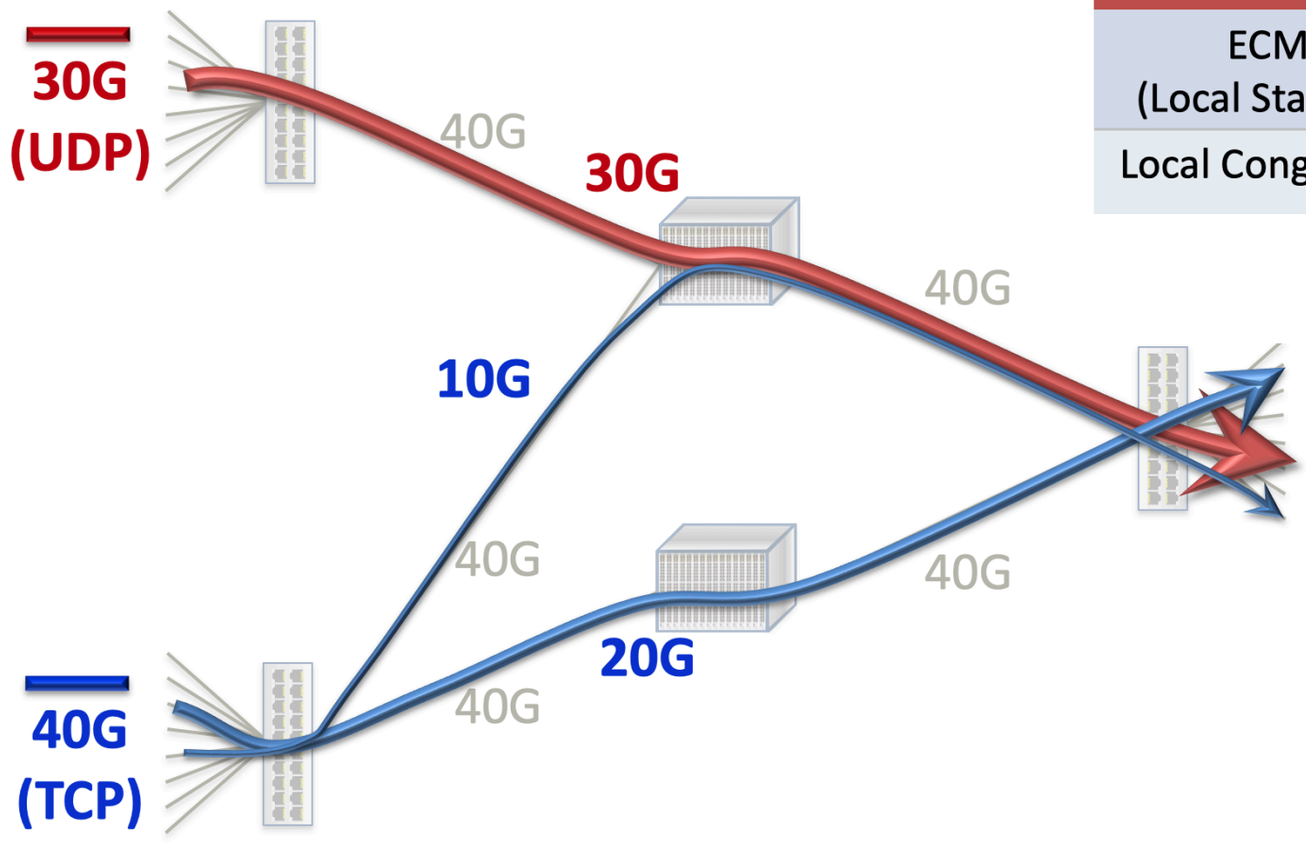
congestion-awareness

~~agnostic~~

local, at the node-level

global, at the network-level

Dealing with Asymmetry: ECMP

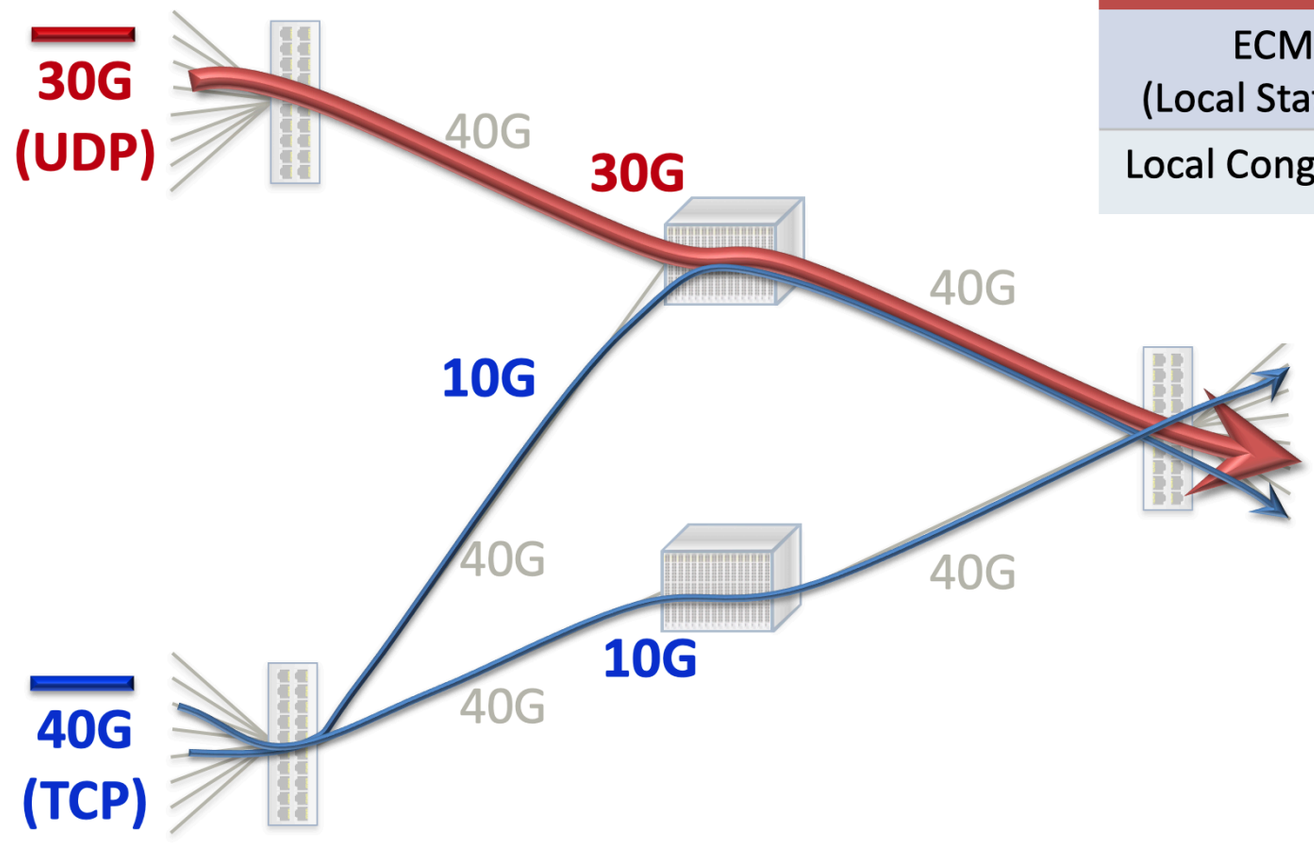


Scheme	Thrput
ECMP (Local Stateless)	**60G**
Local Cong-Aware	

8

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**

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

load-balancing decision…

granularity

~~flow~~

flowlet

~~packet~~

congestion-awareness

~~agnostic~~

~~local~~, at the node-level

handling asymmetry mandates non-local knowledge

global, at the network-level

load-balancing decision…

granularity

~~flow~~

flowlet

~~packet~~

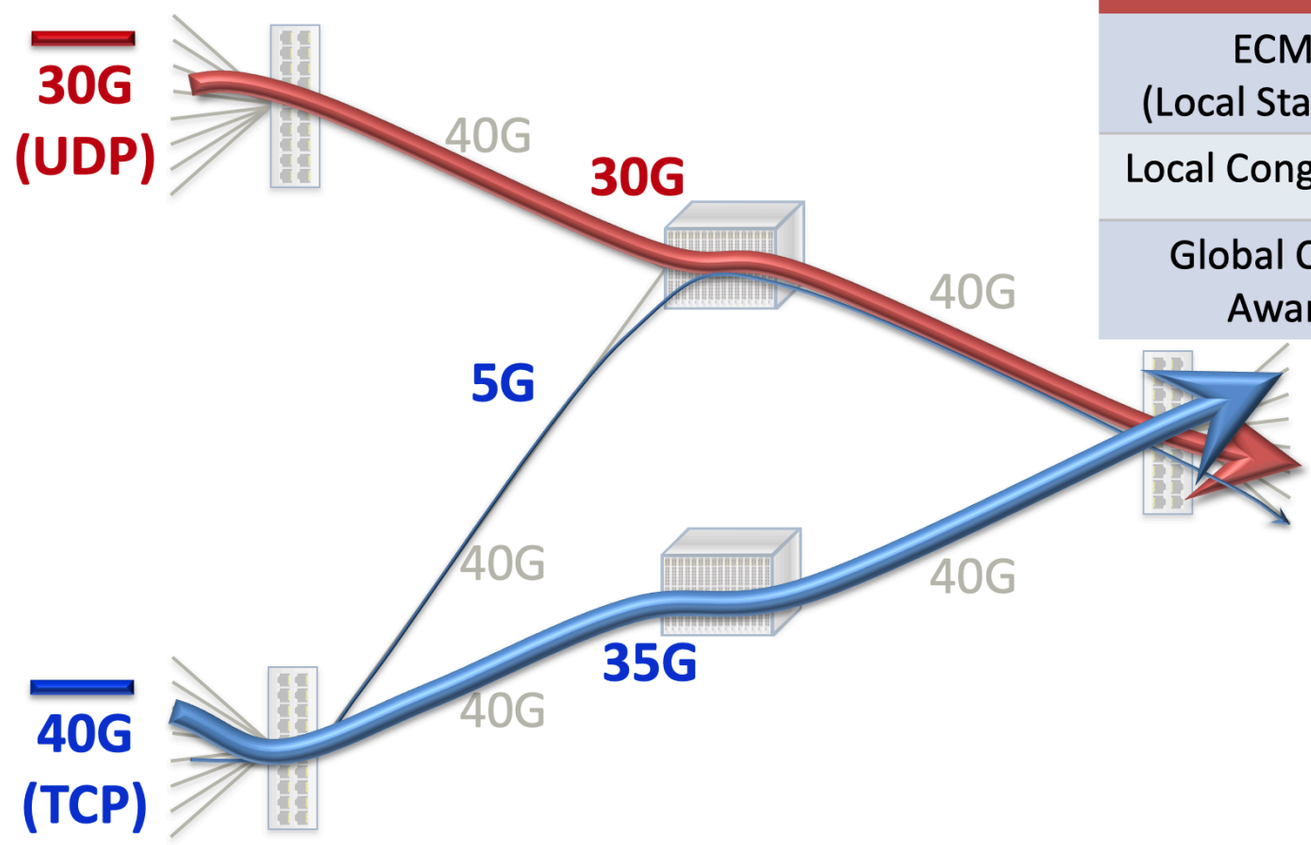
congestion-awareness

~~agnostic~~

~~local, at the node-level~~

global, at the network-level

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

load-balancing decision…

granularity

~~flow~~

flowlet **best pick!**

~~packet~~

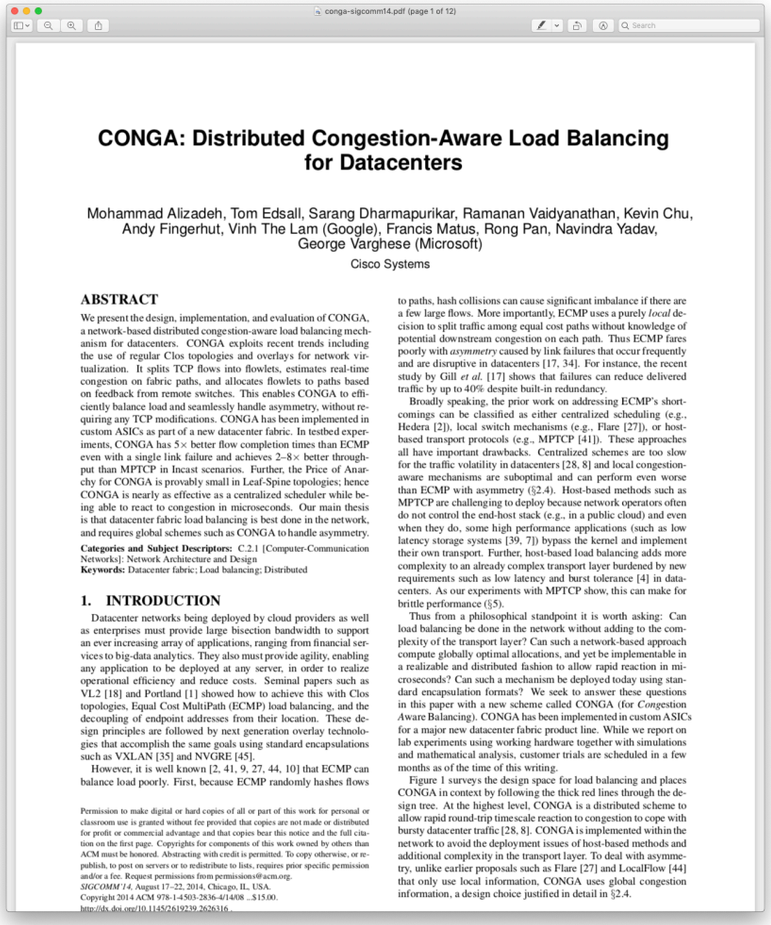
congestion-awareness

~~agnostic~~

~~local, at the node-level~~

global **best pick!**

How do we implement flowlet-based, congestion-aware load balancing?



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).

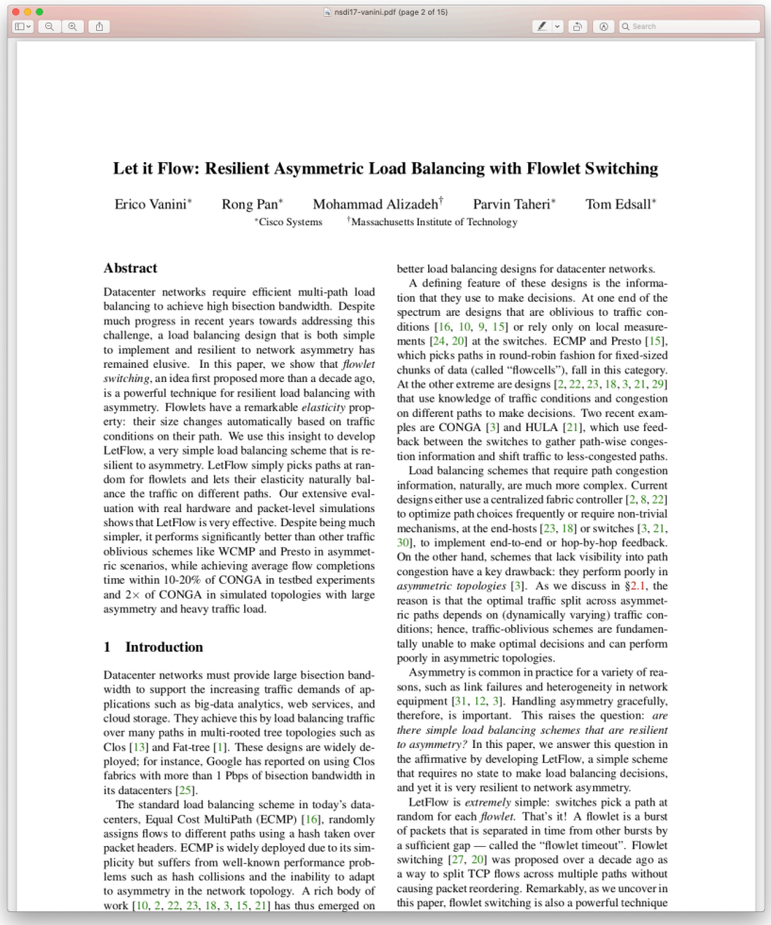
Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow 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 *Con*gestion Aware 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.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626316

CONGA [SIGCOMM'14]



nsdi17-vanini.pdf (page 2 of 15)

Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching

Erico Vanini* Rong Pan* Mohammad Alizadeh† Parvin Taheri* Tom Edsall*

*Cisco Systems †Massachusetts Institute of Technology

Abstract

Datacenter networks require efficient multi-path load balancing to achieve high bisection bandwidth. Despite much progress in recent years towards addressing this challenge, a load balancing design that is both simple to implement and resilient to network asymmetry has remained elusive. In this paper, we show that *flowlet switching*, an idea first proposed more than a decade ago, is a powerful technique for resilient load balancing with asymmetry. Flowlets have a remarkable *elasticity* property: their size changes automatically based on traffic conditions on their path. We use this insight to develop LetFlow, a very simple load balancing scheme that is resilient to asymmetry. LetFlow simply picks paths at random for flowlets and lets their elasticity naturally balance the traffic on different paths. Our extensive evaluation with real hardware and packet-level simulations shows that LetFlow is very effective. Despite being much simpler, it performs significantly better than other traffic oblivious schemes like WCMP and Presto in asymmetric scenarios, while achieving average flow completions time within 10-20% of CONGA in testbed experiments and 2× of CONGA in simulated topologies with large asymmetry and heavy traffic load.

1 Introduction

Datacenter networks must provide large bisection bandwidth to support the increasing traffic demands of applications such as big-data analytics, web services, and cloud storage. They achieve this by load balancing traffic over many paths in multi-rooted tree topologies such as Clos [13] and Fat-tree [1]. These designs are widely deployed; for instance, Google has reported on using Clos fabrics with more than 1 Pbps of bisection bandwidth in its datacenters [25].

The standard load balancing scheme in today's datacenters, Equal Cost MultiPath (ECMP) [16], randomly assigns flows to different paths using a hash taken over packet headers. ECMP is widely deployed due to its simplicity but suffers from well-known performance problems such as hash collisions and the inability to adapt to asymmetry in the network topology. A rich body of work [10, 2, 22, 23, 18, 3, 15, 21] has thus emerged on better load balancing designs for datacenter networks.

A defining feature of these designs is the information that they use to make decisions. At one end of the spectrum are designs that are oblivious to traffic conditions [16, 10, 9, 15] or rely only on local measurements [24, 20] at the switches. ECMP and Presto [15], which picks paths in round-robin fashion for fixed-sized chunks of data (called "flowcells"), fall in this category. At the other extreme are designs [2, 22, 23, 18, 3, 21, 29] that use knowledge of traffic conditions and congestion on different paths to make decisions. Two recent examples are CONGA [3] and HULA [21], which use feedback between the switches to gather path-wise congestion information and shift traffic to less-congested paths.

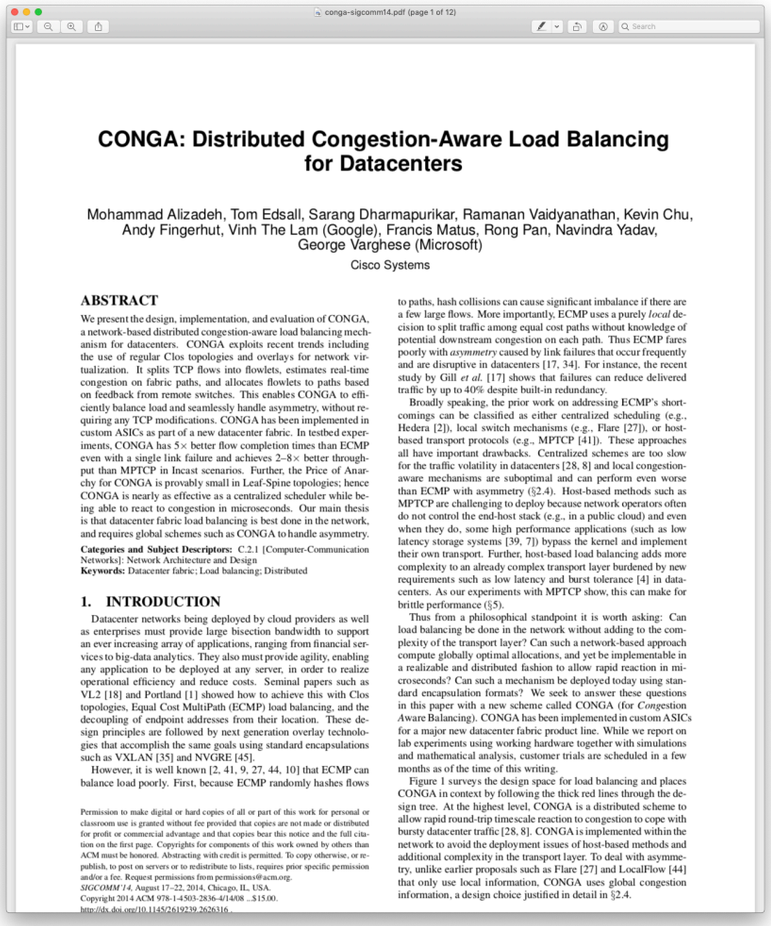
Load balancing schemes that require path congestion information, naturally, are much more complex. Current designs either use a centralized fabric controller [2, 8, 22] to optimize path choices frequently or require non-trivial mechanisms, at the end-hosts [23, 18] or switches [3, 21, 30], to implement end-to-end or hop-by-hop feedback. On the other hand, schemes that lack visibility into path congestion have a key drawback: they perform poorly in *asymmetric topologies* [3]. As we discuss in §2.1, the reason is that the optimal traffic split across asymmetric paths depends on (dynamically varying) traffic conditions; hence, traffic-oblivious schemes are fundamentally unable to make optimal decisions and can perform poorly in asymmetric topologies.

Asymmetry is common in practice for a variety of reasons, such as link failures and heterogeneity in network equipment [31, 12, 3]. Handling asymmetry gracefully, therefore, is important. This raises the question: *are there simple load balancing schemes that are resilient to asymmetry?* In this paper, we answer this question in the affirmative by developing LetFlow, a simple scheme that requires no state to make load balancing decisions, and yet it is very resilient to network asymmetry.

LetFlow is *extremely* simple: switches pick a path at random for each *flowlet*. That's it! A flowlet is a burst of packets that is separated in time from other bursts by a sufficient gap — called the "flowlet timeout". Flowlet switching [27, 20] was proposed over a decade ago as a way to split TCP flows across multiple paths without causing packet reordering. Remarkably, as we uncover in this paper, flowlet switching is also a powerful technique

LetFlow [NSDI'17]

How do we implement flowlet-based, congestion-aware load balancing?



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 *Con*gestion *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.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626316 .

CONGA [SIGCOMM'14]

nsdi17-vanini.pdf (page 2 of 15)

Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching

Erico Vanini* Rong Pan* Mohammad Alizadeh† Parvin Taheri* Tom Edsall*
*Cisco Systems †Massachusetts Institute of Technology

Abstract

Datacenter networks require efficient multi-path load balancing to achieve high bisection bandwidth. Despite much progress in recent years towards addressing this challenge, a load balancing design that is both simple to implement and resilient to network asymmetry has remained elusive. In this paper, we show that *flowlet switching*, an idea first proposed more than a decade ago, is a powerful technique for resilient load balancing with asymmetry. Flowlets have a remarkable *elasticity* property: their size changes automatically based on traffic conditions on their path. We use this insight to develop LetFlow, a very simple load balancing scheme that is resilient to asymmetry. LetFlow simply picks paths at random for flowlets and lets their elasticity naturally balance the traffic on different paths. Our extensive evaluation with real hardware and packet-level simulations shows that LetFlow is very effective. Despite being much simpler, it performs significantly better than other traffic oblivious schemes like WCMP and Presto in asymmetric scenarios, while achieving average flow completions time within 10-20% of CONGA in testbed experiments and 2× of CONGA in simulated topologies with large asymmetry and heavy traffic load.

1 Introduction

Datacenter networks must provide large bisection bandwidth to support the increasing traffic demands of applications such as big-data analytics, web services, and cloud storage. They achieve this by load balancing traffic over many paths in multi-rooted tree topologies such as Clos [13] and Fat-tree [1]. These designs are widely deployed; for instance, Google has reported on using Clos fabrics with more than 1 Pbps of bisection bandwidth in its datacenters [25].

The standard load balancing scheme in today's datacenters, Equal Cost MultiPath (ECMP) [16], randomly assigns flows to different paths using a hash taken over packet headers. ECMP is widely deployed due to its simplicity but suffers from well-known performance problems such as hash collisions and the inability to adapt to asymmetry in the network topology. A rich body of work [10, 2, 22, 23, 18, 3, 15, 21] has thus emerged on better load balancing designs for datacenter networks.

A defining feature of these designs is the information that they use to make decisions. At one end of the spectrum are designs that are oblivious to traffic conditions [16, 10, 9, 15] or rely only on local measurements [24, 20] at the switches. ECMP and Presto [15], which picks paths in round-robin fashion for fixed-sized chunks of data (called "flowcells"), fall in this category. At the other extreme are designs [2, 22, 23, 18, 3, 21, 29] that use knowledge of traffic conditions and congestion on different paths to make decisions. Two recent examples are CONGA [3] and HULA [21], which use feedback between the switches to gather path-wise congestion information and shift traffic to less-congested paths.

Load balancing schemes that require path congestion information, naturally, are much more complex. Current designs either use a centralized fabric controller [2, 8, 22] to optimize path choices frequently or require non-trivial mechanisms, at the end-hosts [23, 18] or switches [3, 21, 30], to implement end-to-end or hop-by-hop feedback. On the other hand, schemes that lack visibility into path congestion have a key drawback: they perform poorly in *asymmetric topologies* [3]. As we discuss in §2.1, the reason is that the optimal traffic split across asymmetric paths depends on (dynamically varying) traffic conditions; hence, traffic-oblivious schemes are fundamentally unable to make optimal decisions and can perform poorly in asymmetric topologies.

Asymmetry is common in practice for a variety of reasons, such as link failures and heterogeneity in network equipment [31, 12, 3]. Handling asymmetry gracefully, therefore, is important. This raises the question: *are there simple load balancing schemes that are resilient to asymmetry?* In this paper, we answer this question in the affirmative by developing LetFlow, a simple scheme that requires no state to make load balancing decisions, and yet it is very resilient to network asymmetry.

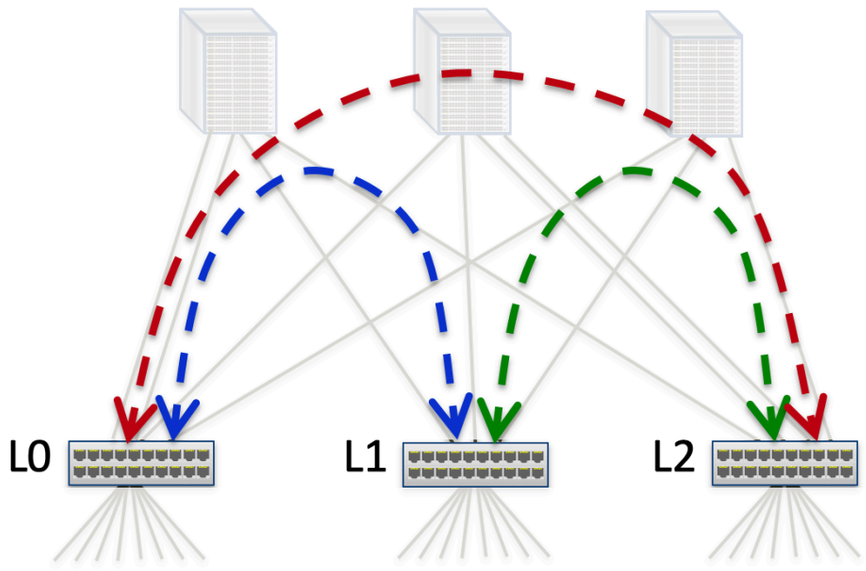
LetFlow is *extremely* simple: switches pick a path at random for each *flowlet*. That's it! A flowlet is a burst of packets that is separated in time from other bursts by a sufficient gap — called the "flowlet timeout". Flowlet switching [27, 20] was proposed over a decade ago as a way to split TCP flows across multiple paths without causing packet reordering. Remarkably, as we uncover in this paper, flowlet switching is also a powerful technique

LetFlow [NSDI'17]

CONGA in 1 Slide

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

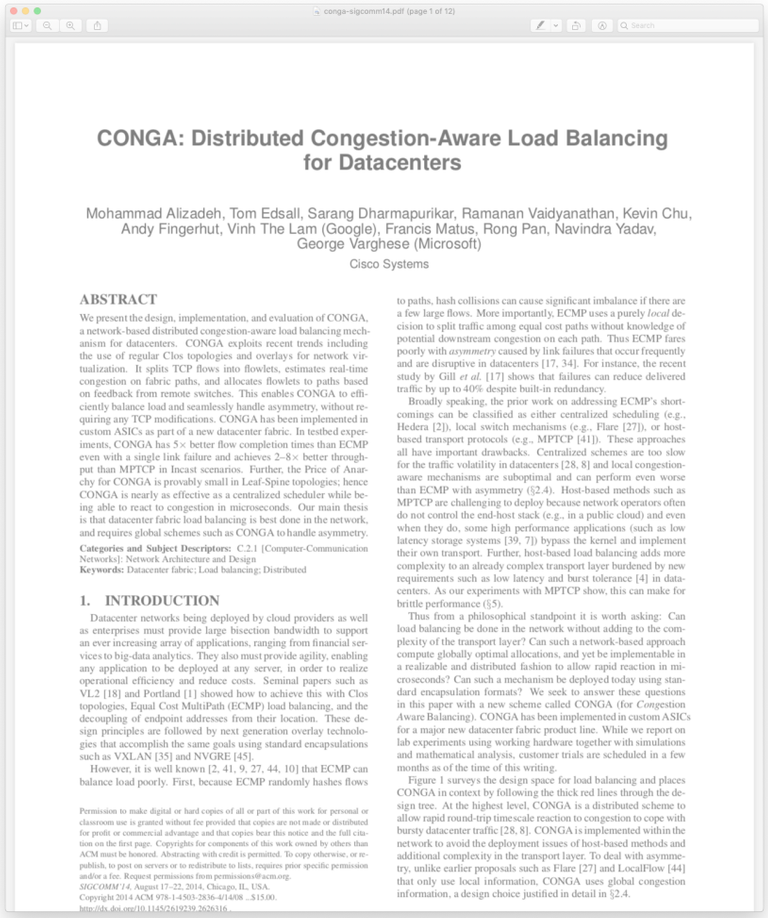
1. Use greedy decisions to minimize bottleneck util



Fast feedback loops between leaf switches, directly in dataplane

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

How do we implement flowlet-based, congestion-aware load balancing?



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).

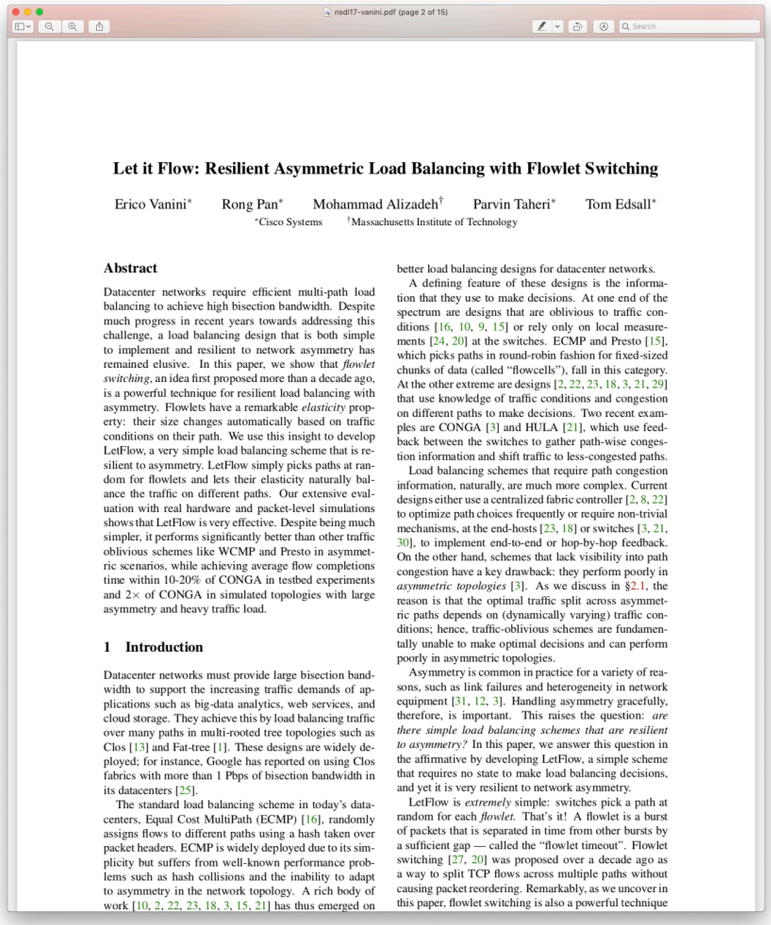
Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow 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 *Con*gestion Aware 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.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626316.

CONGA [SIGCOMM'14]



Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching

Erico Vanini* Rong Pan* Mohammad Alizadeh† Parvin Taheri* Tom Edsall*
*Cisco Systems †Massachusetts Institute of Technology

Abstract

Datacenter networks require efficient multi-path load balancing to achieve high bisection bandwidth. Despite much progress in recent years towards addressing this challenge, a load balancing design that is both simple to implement and resilient to network asymmetry has remained elusive. In this paper, we show that *flowlet switching*, an idea first proposed more than a decade ago, is a powerful technique for resilient load balancing with asymmetry. Flowlets have a remarkable *elasticity* property: their size changes automatically based on traffic conditions on their path. We use this insight to develop LetFlow, a very simple load balancing scheme that is resilient to asymmetry. LetFlow simply picks paths at random for flowlets and lets their elasticity naturally balance the traffic on different paths. Our extensive evaluation with real hardware and packet-level simulations shows that LetFlow is very effective. Despite being much simpler, it performs significantly better than other traffic oblivious schemes like WCMP and Presto in asymmetric scenarios, while achieving average flow completions time within 10-20% of CONGA in testbed experiments and 2× of CONGA in simulated topologies with large asymmetry and heavy traffic load.

1 Introduction

Datacenter networks must provide large bisection bandwidth to support the increasing traffic demands of applications such as big-data analytics, web services, and cloud storage. They achieve this by load balancing traffic over many paths in multi-rooted tree topologies such as Clos [13] and Fat-tree [1]. These designs are widely deployed; for instance, Google has reported on using Clos fabrics with more than 1 Pbps of bisection bandwidth in its datacenters [25].

The standard load balancing scheme in today's datacenters, Equal Cost MultiPath (ECMP) [16], randomly assigns flows to different paths using a hash taken over packet headers. ECMP is widely deployed due to its simplicity but suffers from well-known performance problems such as hash collisions and the inability to adapt to asymmetry in the network topology. A rich body of work [10, 2, 22, 23, 18, 3, 15, 21] has thus emerged on better load balancing designs for datacenter networks.

A defining feature of these designs is the information that they use to make decisions. At one end of the spectrum are designs that are oblivious to traffic conditions [16, 10, 9, 15] or rely only on local measurements [24, 20] at the switches. ECMP and Presto [15], which picks paths in round-robin fashion for fixed-sized chunks of data (called "flowcells"), fall in this category. At the other extreme are designs [2, 22, 23, 18, 3, 21, 29] that use knowledge of traffic conditions and congestion on different paths to make decisions. Two recent examples are CONGA [3] and HULA [21], which use feedback between the switches to gather path-wise congestion information and shift traffic to less-congested paths.

Load balancing schemes that require path congestion information, naturally, are much more complex. Current designs either use a centralized fabric controller [2, 8, 22] to optimize path choices frequently or require non-trivial mechanisms, at the end-hosts [23, 18] or switches [3, 21, 30], to implement end-to-end or hop-by-hop feedback. On the other hand, schemes that lack visibility into path congestion have a key drawback: they perform poorly in *asymmetric topologies* [3]. As we discuss in §2.1, the reason is that the optimal traffic split across asymmetric paths depends on (dynamically varying) traffic conditions; hence, traffic-oblivious schemes are fundamentally unable to make optimal decisions and can perform poorly in asymmetric topologies.

Asymmetry is common in practice for a variety of reasons, such as link failures and heterogeneity in network equipment [31, 12, 3]. Handling asymmetry gracefully, therefore, is important. This raises the question: *are there simple load balancing schemes that are resilient to asymmetry?* In this paper, we answer this question in the affirmative by developing LetFlow, a simple scheme that requires no state to make load balancing decisions, and yet it is very resilient to network asymmetry.

LetFlow is *extremely* simple: switches pick a path at random for each *flowlet*. That's it! A flowlet is a burst of packets that is separated in time from other bursts by a sufficient gap — called the "flowlet timeout". Flowlet switching [27, 20] was proposed over a decade ago as a way to split TCP flows across multiple paths without causing packet reordering. Remarkably, as we uncover in this paper, flowlet switching is also a powerful technique

LetFlow [NSDI'17]

Existing Load Balancing Schemes

Congestion-aware decisions: complex

- Measure and feed back congestion in real time
- CONGA, Hedera, HULA, MPTCP, FlowBender,…

Congestion-oblivious decisions: simple

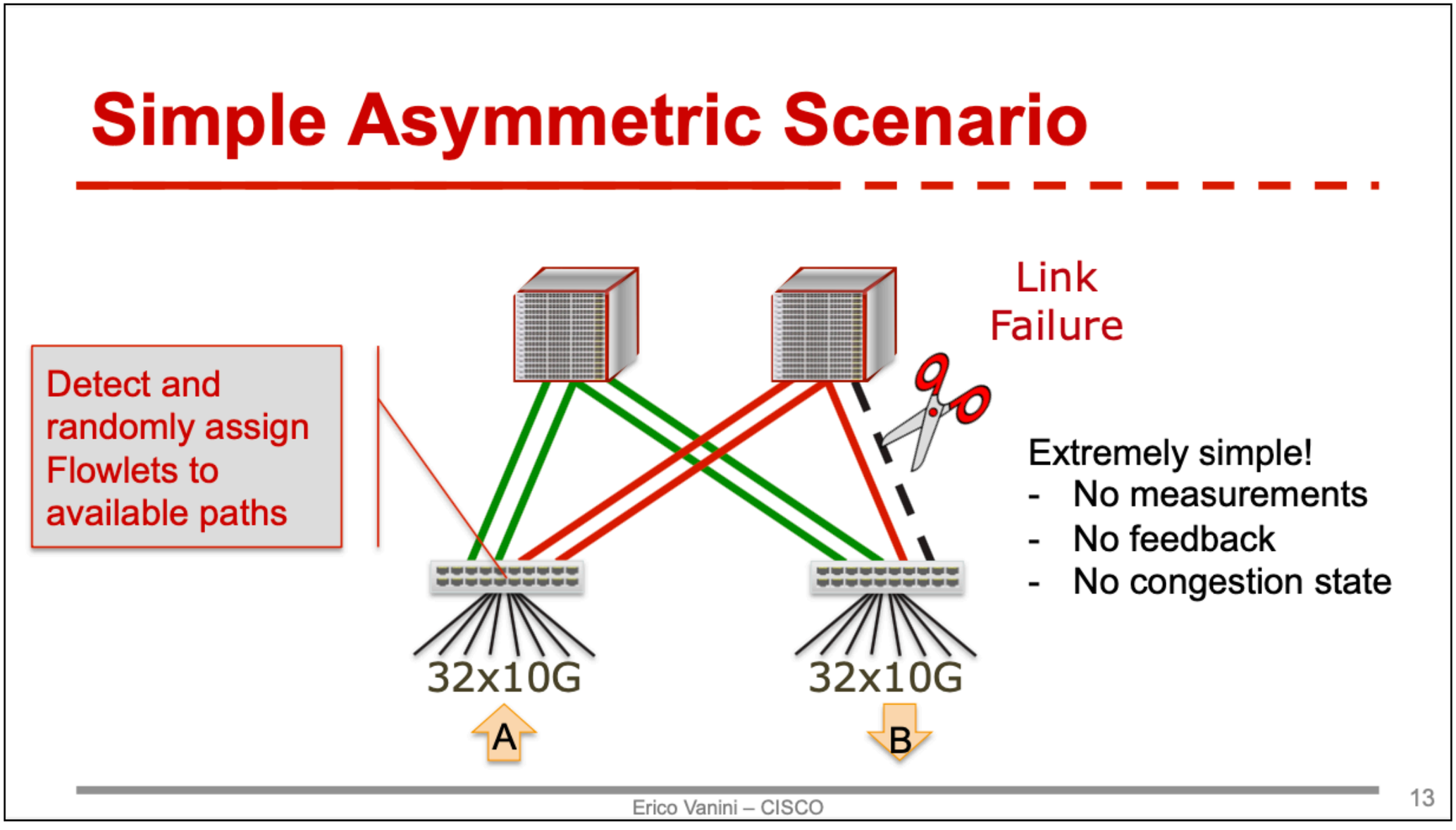
- Random, round robin, hashing decision process
- ECMP, WCMP, Packet-Spray, Presto,…

Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

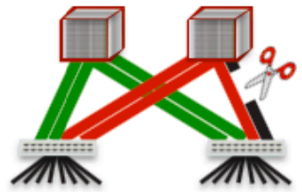
LetFlow:

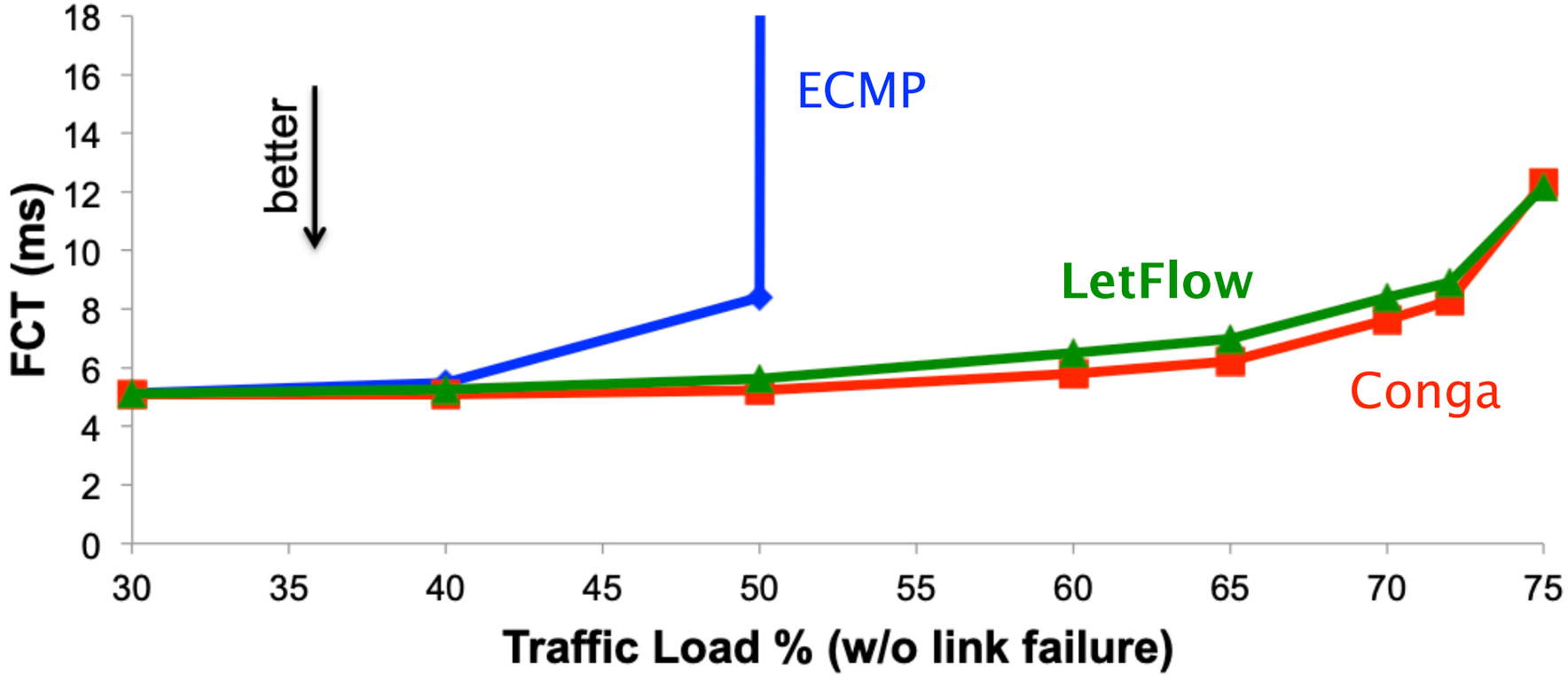
Randomly assign flowlets to available paths. ***That's it!***



Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

LetFlow





Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

Flowlets are Elastic

- Flowlets change size based on congestion on the path
- Uncongested path → larger flowlets
- Congested path → smaller flowlets

→ Flowlet sizes **implicitly** encode path congestion information

… this determines the amount of traffic on each path – not just load balancing decisions

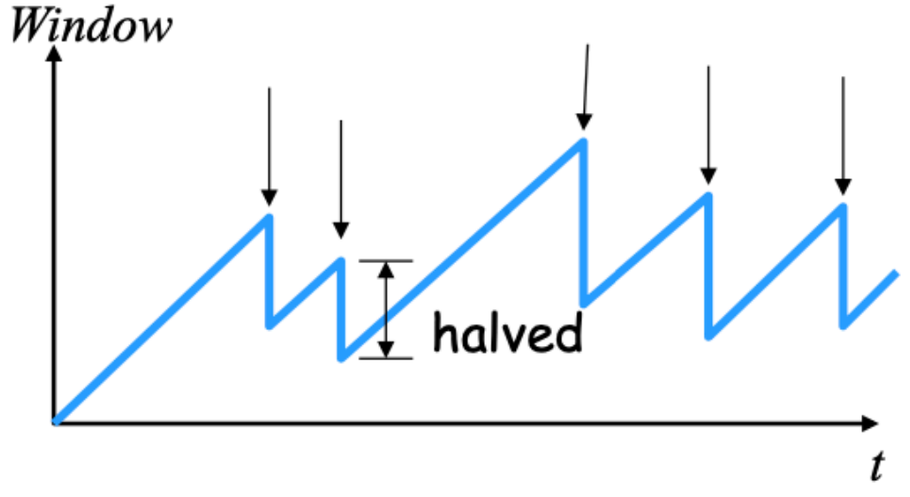
LetFlow *is* congestion-aware, despite simple random decisions

Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

Why Are Flowlets Elastic?

- Because of congestion control (e.g., TCP)

- A flowlet gap occurs on
  - Window cuts (Loss/ECN)
  - Latency spikes (ACK clocking)



Source: Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching, Erico Vanini et al., 2017

Advanced Topics in Communication Networks

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich

Tue 4 Oct 2022