Advanced Topics in Communication Networks

Internet Routing and Forwarding

Laurent Vanbever

nsg.ee.ethz.ch

13 Oct 2020

Lecture starts at **14:15**

Materials inspired and/or coming from Olivier Bonaventure

Last week on

Advanced Topics in Communication Networks

https://padlet.com/romainjacob42/qxhdeqy7cd6nm05t

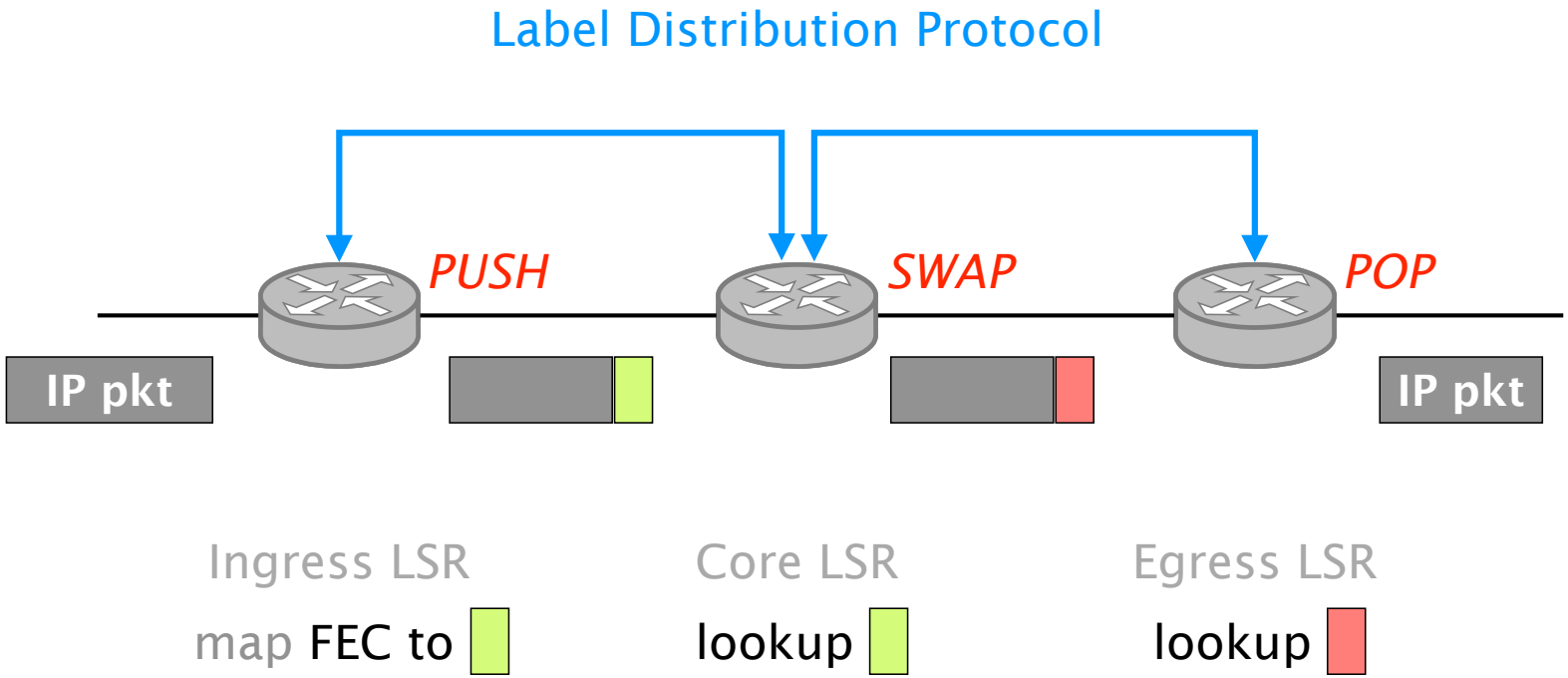
1. How should we compare a software-programmable (P4) switch with more traditional L2/L3 switches or L3 routers?
2. Can I also do NAT using P4?
3. How can I make a P4 program modular?
4. Are there libraries of classical P4 programs?

5. How does the ternary match work?
6. Which operations may slow down the program and hence slow down switching?

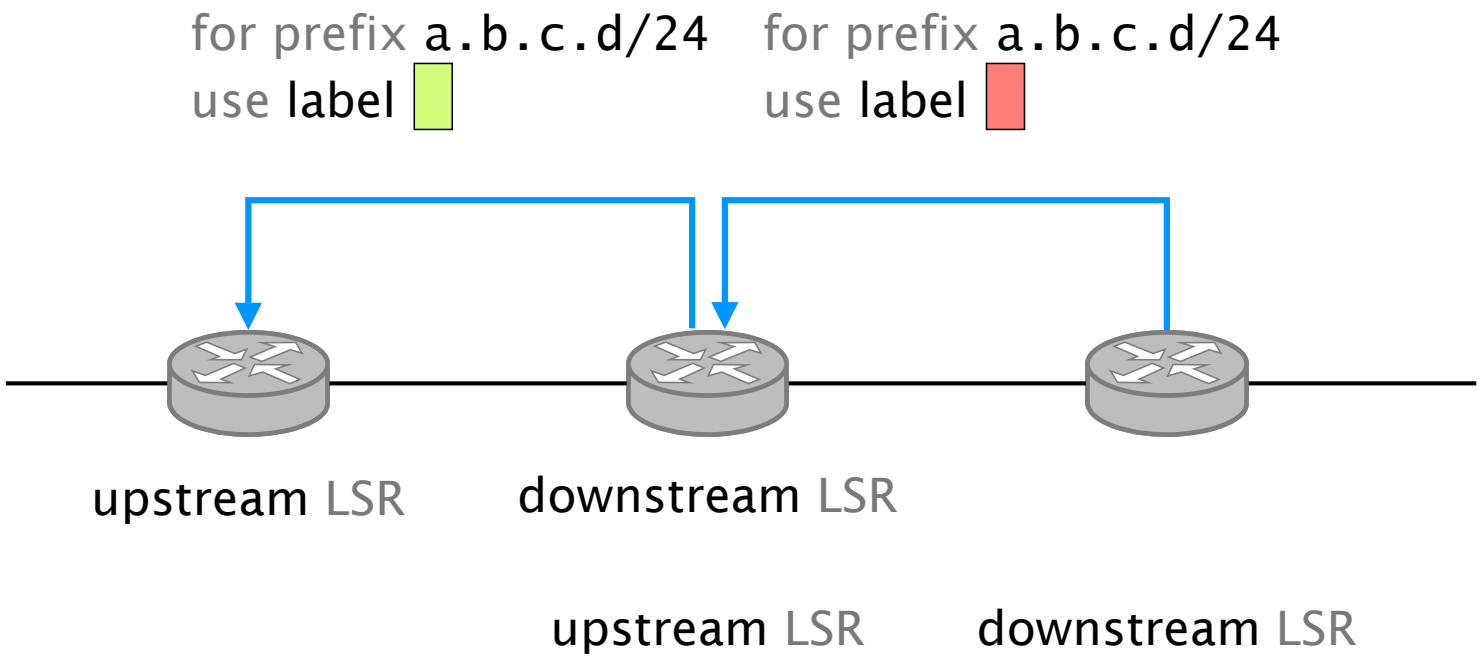
7. What is the point of penultimate popping?
8. How do MPLS routers deal with MTU to fit extra MPLS headers?

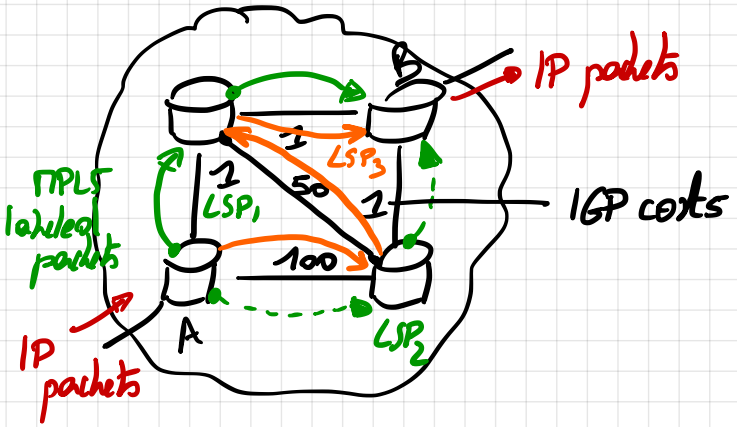
Multiprotocol Label Switching

"IP meets virtual circuits"



Label Distribution Protocol





* How do we build/signal these LSPs, along wide which paths, and for which FECs?
  → Label Distribution Protocol

last week

LDP

- Build LSPs following the IGP shortest-path trees,
- for IGP-learned prefixes.

RSVP-TE

- Build LSPs following arbitrary paths in the network.
- for arbitrary / user-provided FECs.

this week

This week on

Advanced Topics in Communication Networks

Advanced
Load Balancing

How can we do better than ECMP?
(using smarter data planes)

MPLS-based
Traffic Engineering

RSVP-TE
(at long last)

Advanced
Load Balancing

MPLS-based
Traffic Engineering

How can we do better than ECMP?

(using smarter data planes)

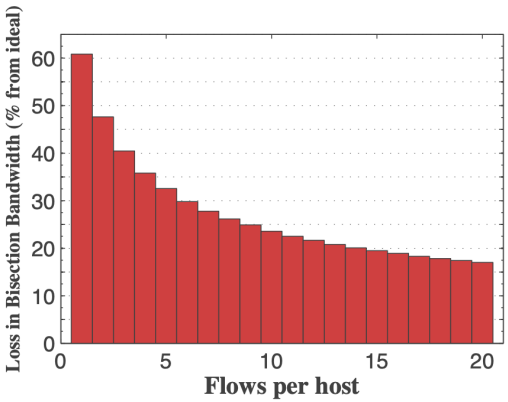
How can we fight ECMP main two shortcomings:

hash collisions and asymmetry

How can we fight ECMP main two shortcomings: hash collisions and asymmetry

ECMP randomly hashes flows to paths (see last week)

Hash collisions can cause significant imbalance if there are few large flows



How can we fight ECMP main two shortcomings:

hash collisions and asymmetry

ECMP uses purely local decision to split flows across paths,
without any knowledge of potential downstream congestion

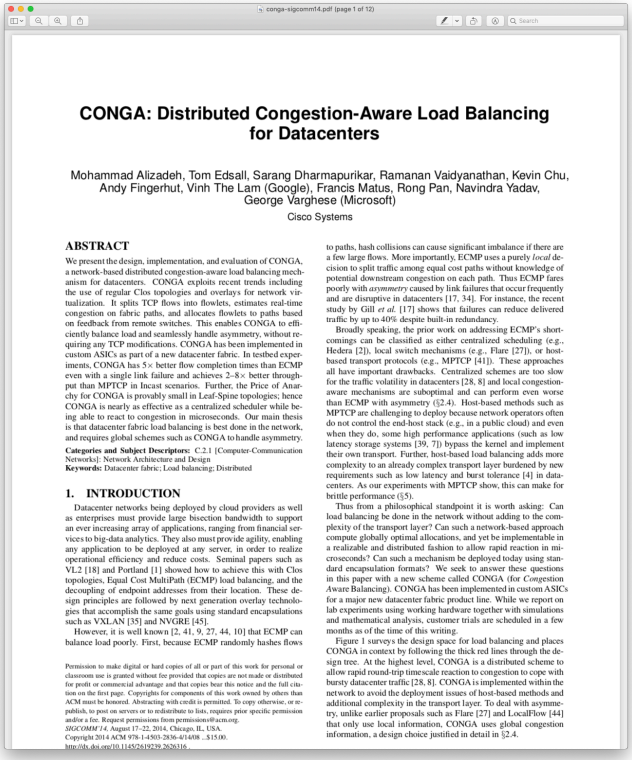
How can we fight ECMP main two shortcomings:
hash collisions and asymmetry

by making…

key insight congestion-aware
load balancing decisions

Let's see together two recent proposals that achieve congestion-aware load balancing

use network-wide feedback



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.

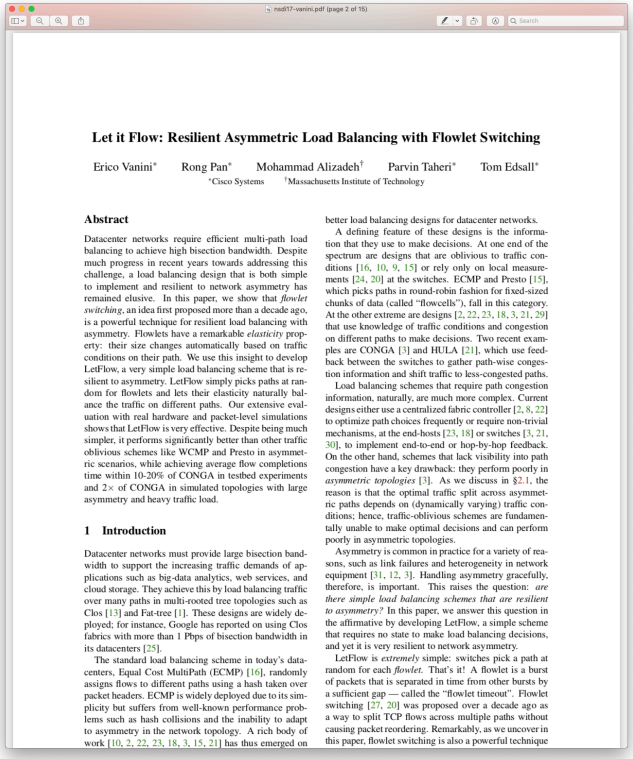
Figure 1 surveys the design space for load balancing and places CONGA in context by following the thick red lines through the design tree. At the highest level, CONGA is a distributed scheme to allow rapid round-trip timescale reaction to congestion to cope with bursty datacenter traffic [28, 8]. CONGA 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]

locally load balance "elastic" entities



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.

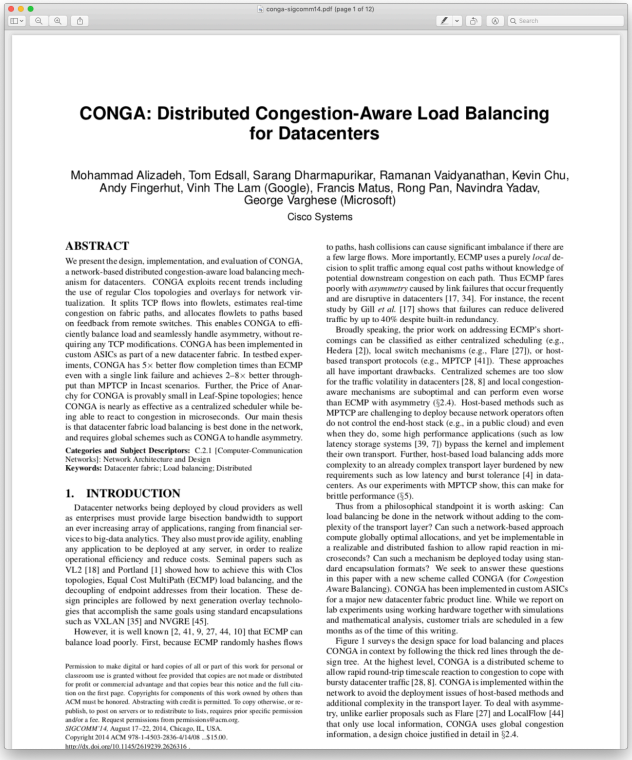
Load balancing schemes that require path congestion information, naturally, are much more complex. Current designs either use a centralized fabric controller [2, 8, 22] to optimize path choices frequently or require non-trivial mechanisms, at the end-hosts [23, 18] or switches [3, 21, 30], to implement 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]

use network-wide feedback



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]

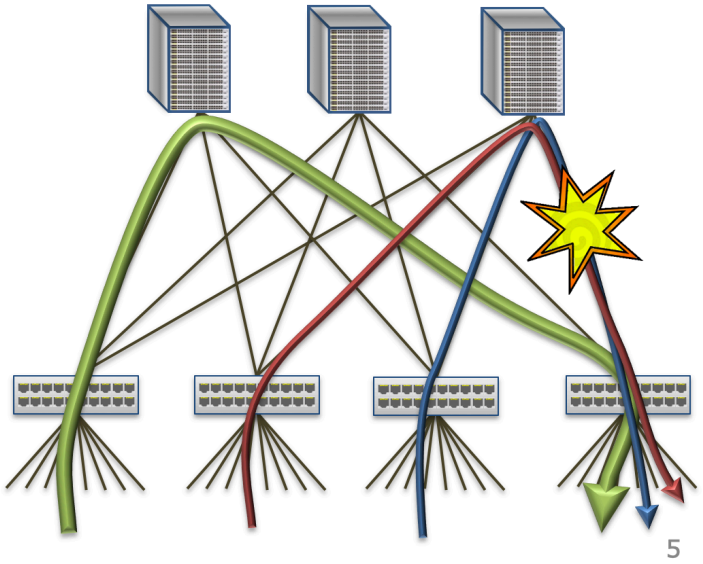
Today: ECMP Load Balancing

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

- Approximates Valiant load balancing
- Preserves packet order

Problems:

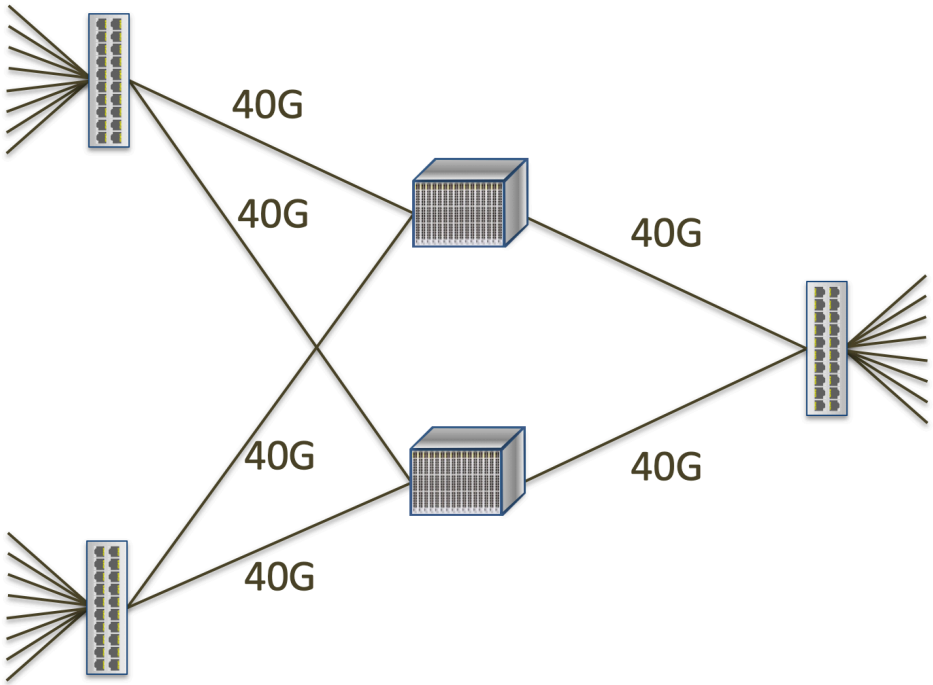
- Hash collisions
  (coarse granularity)
- Local & stateless
  (v. bad with asymmetry
  due to link failures)



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

Dealing with Asymmetry

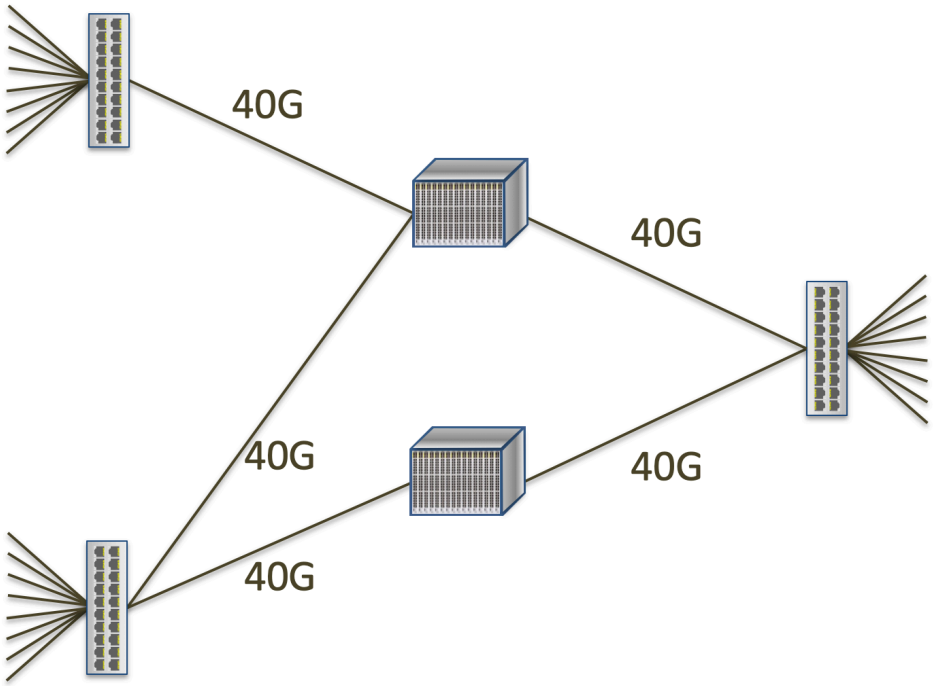
Handling asymmetry needs non-local knowledge



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

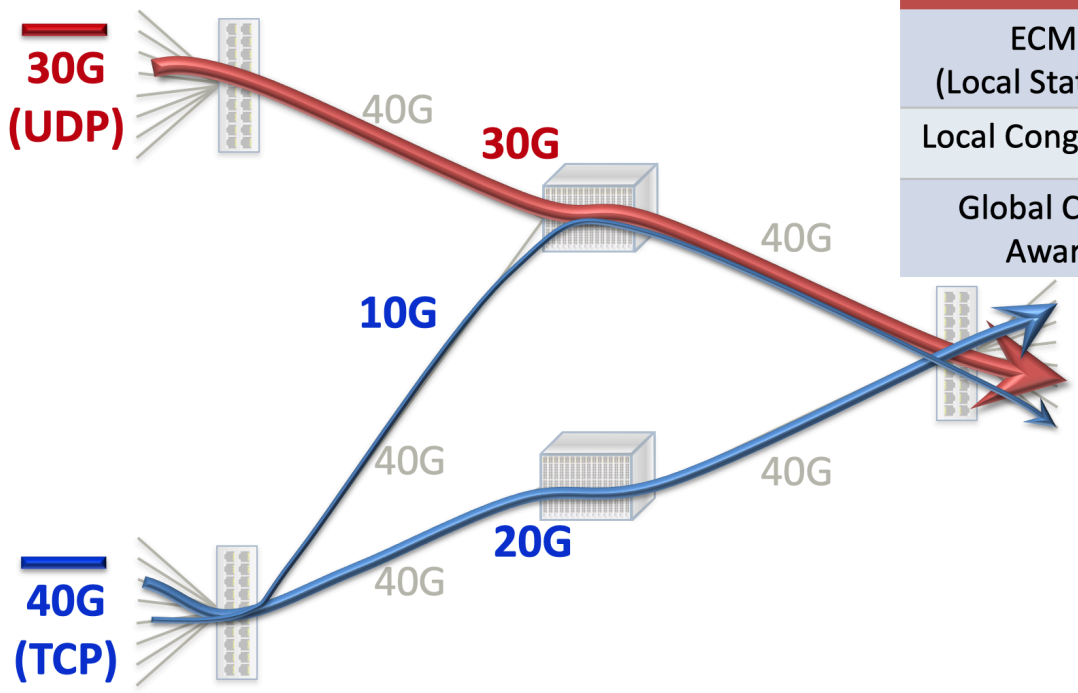
Dealing with Asymmetry

Handling asymmetry needs non-local knowledge



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

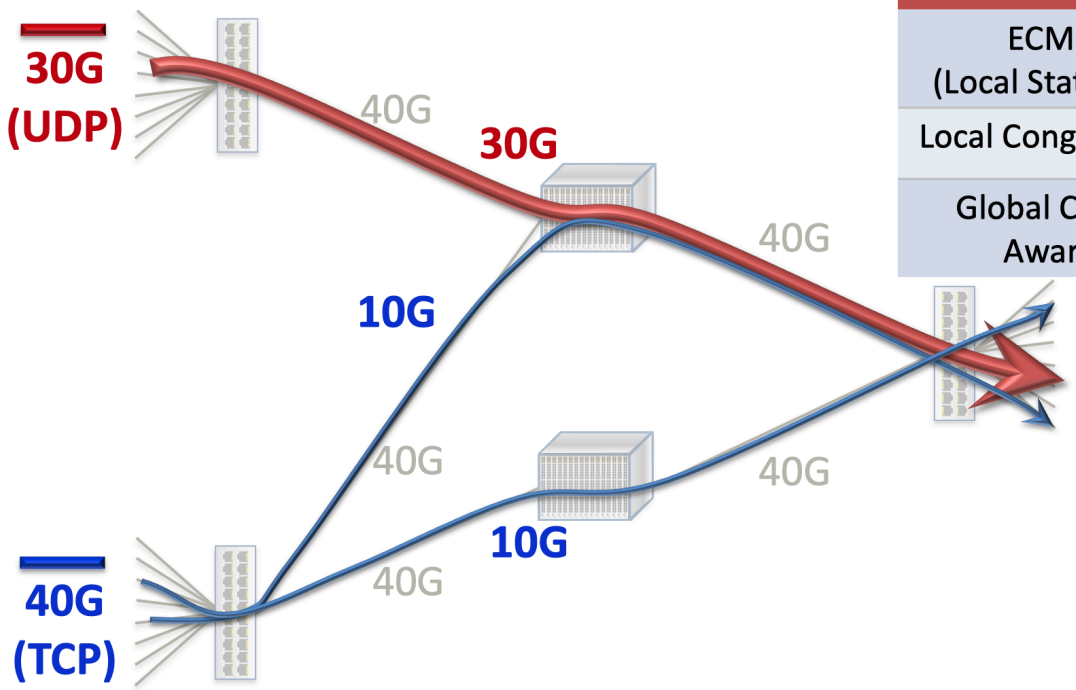
Dealing with Asymmetry: ECMP



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

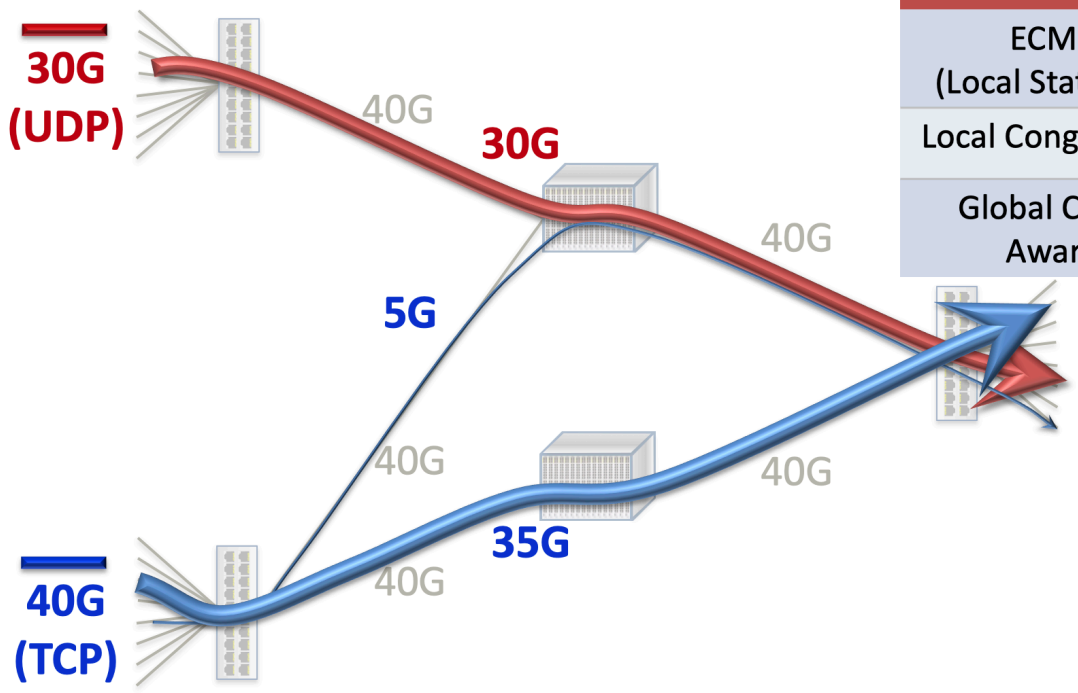
Dealing with Asymmetry: Local Congestion-Aware



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

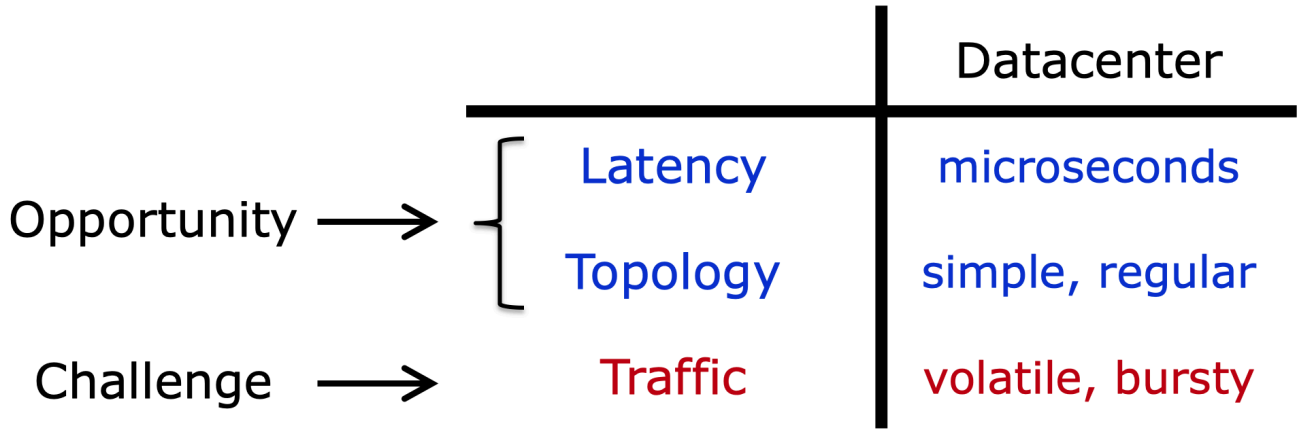
Dealing with Asymmetry: Global Congestion-Aware



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

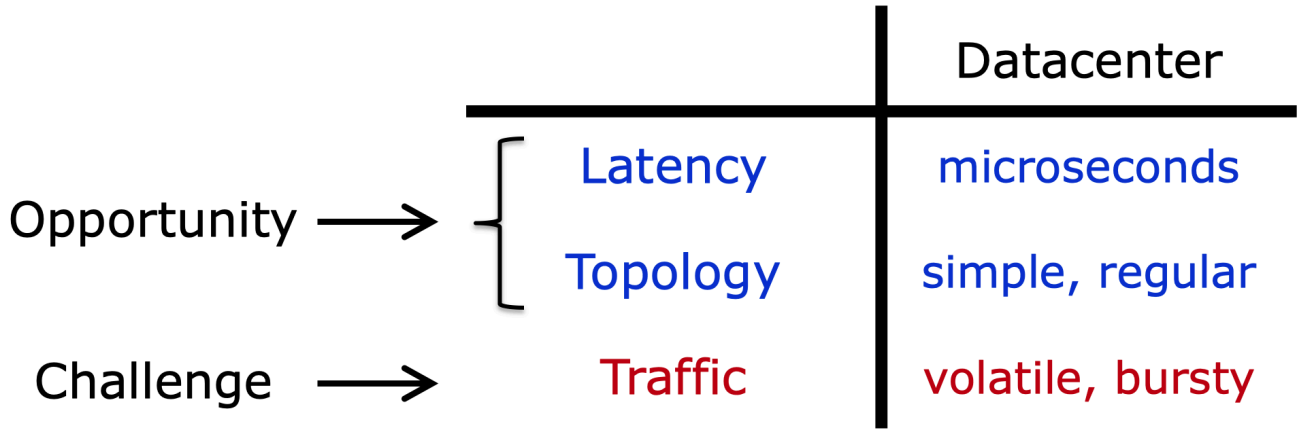
Global Congestion-Awareness (in Datacenters)



11

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

Global Congestion-Awareness (in Datacenters)



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

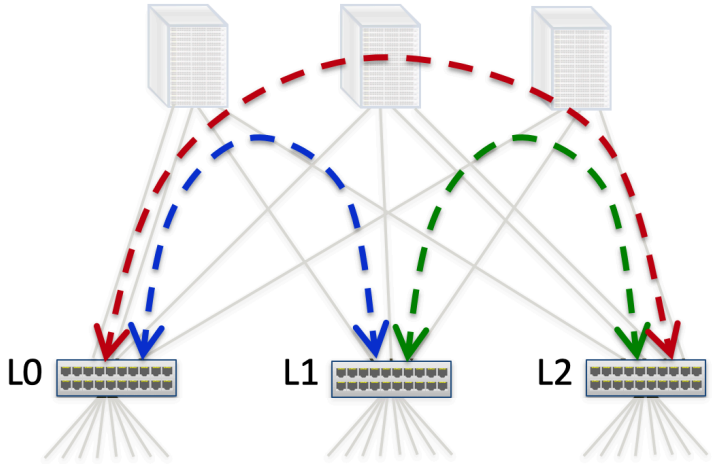
11

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

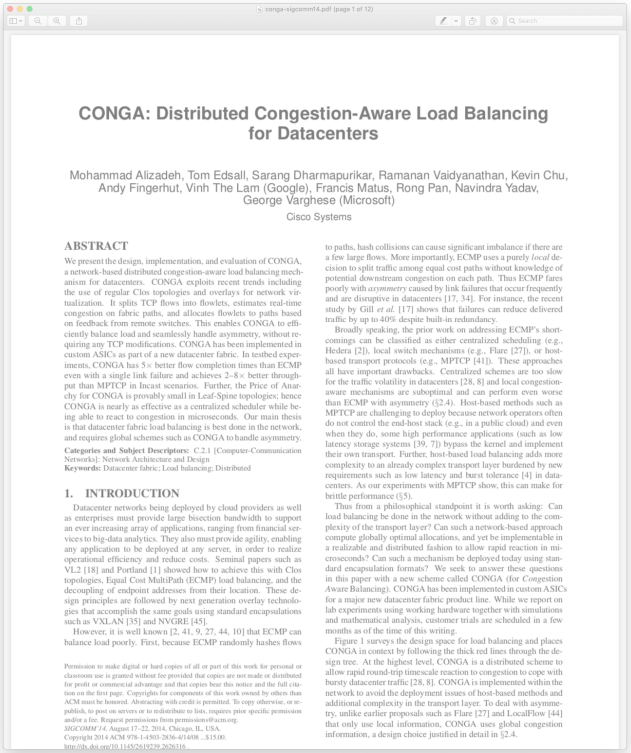


Fast feedback loops between leaf switches, directly in dataplane

12

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

locally load balance "elastic" entities



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).

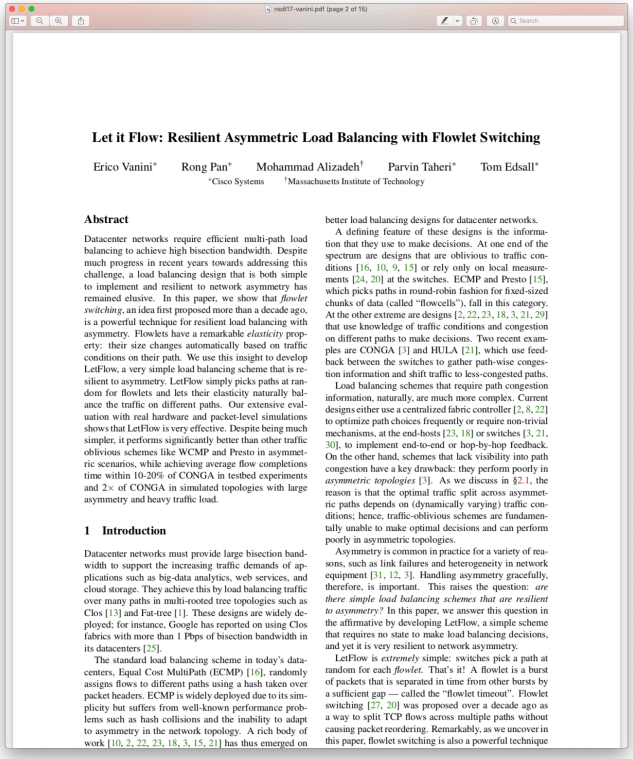
Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow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



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, Vanini et al., 2017

Is there a simple load balancing scheme (with congestion-oblivious decisions) that can handle asymmetry?

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

LetFlow

Simple:

Randomly assign Flowlets to available paths

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, 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, Vanini et al., 2017

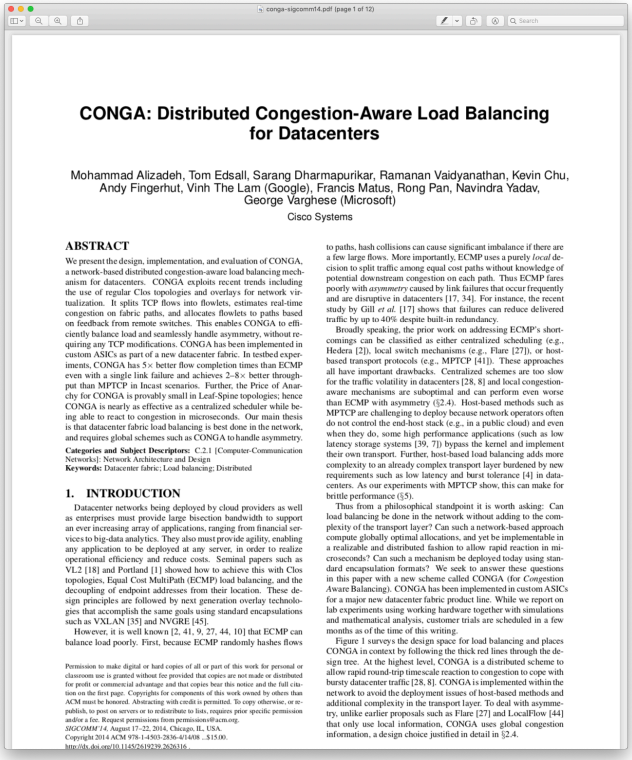
Conclusion

- Flowlet switching is a powerful technique for asymmetric load balancing
- LetFlow: a simple LB mechanism that handles asymmetry
  - Random decisions but implicitly congestion-aware
  - Suitable for standalone switches – does not need feedback
- Letflow is stochastic and reactive in nature
  - Cannot proactively prevent congestion / queue buildup like more sophisticated schemes

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

Both CONGA and LetFlow are implementable in P4

You'll implement LetFlow in today's exercises



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).

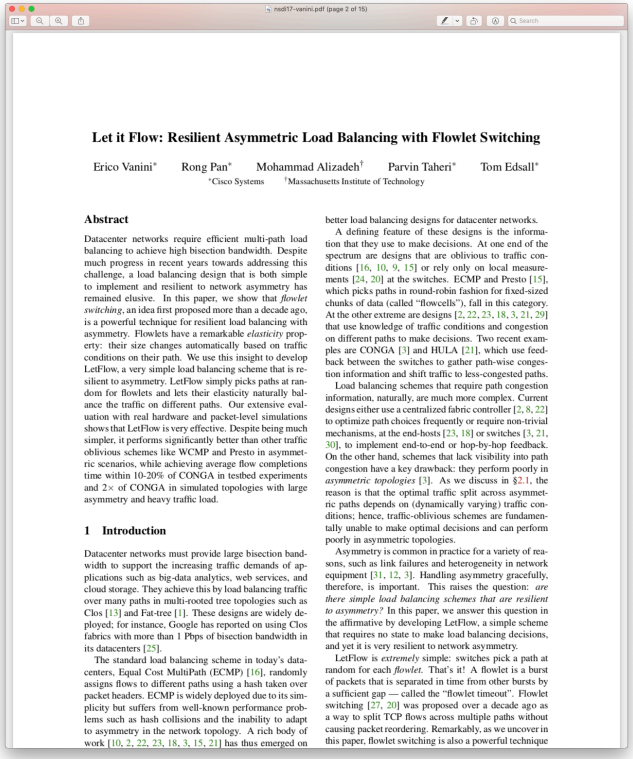
Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow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



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]

Advanced
Load Balancing

MPLS-based
Traffic Engineering

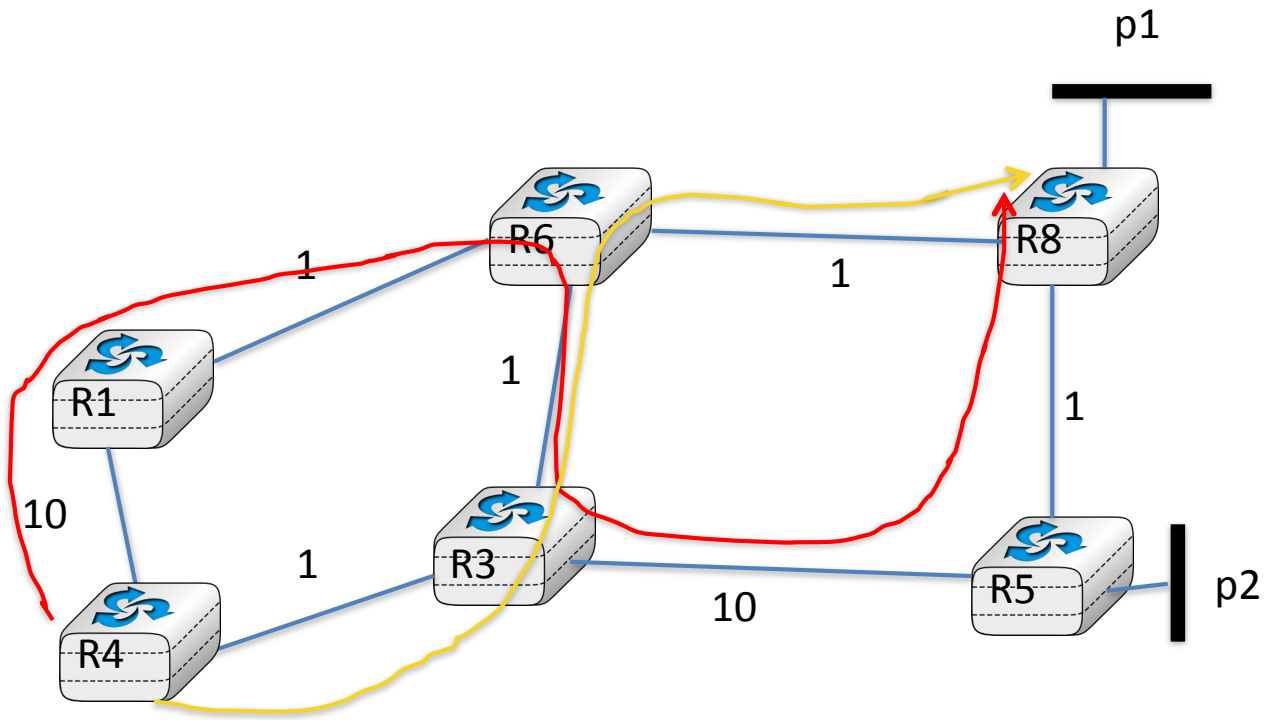
RSVP-TE
(at long last)

MPLS-based Traffic Engineering

- Build a normal IP or IP+MPLS network
  - packet forwarding on shortest path towards destination
- Collect traffic statistics at edge routers and information about link load
  - identify the most congested parts of the network
- Ingress routers establish *LSPs along well chosen paths* to divert large traffic flows away from heavily loaded links

MPLS-based Traffic Engineering

- How to create LSPs along a non-shortest path ?



MPLS-based Traffic Engineering

- Two sub-problems

1. How to find a path through the network that meets the requirements for the LSP?

2. How to signal the LSP along this chosen path?

Typical requirements for an LSP

- Ability to reserve *x* Mbps on all links
- Maximum propagation delay of *y* msec
- Minimise propagation delay
- Ensure that a new LSP does not use the same resources (links, nodes) as an existing one

MPLS TE relies on new link attributes propagated by link state routing protocols

- link type and link id
- local and remote IP addresses
- Traffic Engineering metric
  - additional metric to specify the cost of this link (usually delay)
- maximum bandwidth
  - maximum amount of bandwidth usable on this link
- maximum reservable bandwidth
  - maximum amount of bandwidth that can be reserved by LSPs
- unreserved bandwidth
  - amount of bandwidth that is not yet reserved by LSPs
- resource class/color
  - can be used to specify the type of link
    (e.g. expensive link would be colored in red and cheap links in green)

How does an ingress LSR find a compliant path through the network ?

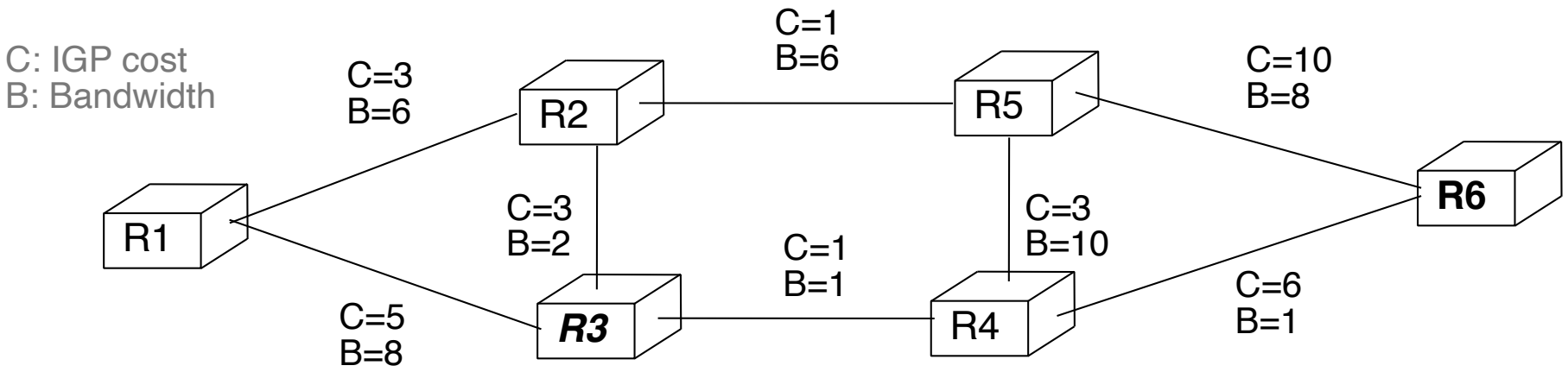
- Path that minimises additive constraint
  - Propagation delay (TE metric)
  - IGP metric
- Dijkstra algorithm
  - Works with only one constraint, in practice either
    - minimize Sum(TE metrics)
    - minimize Sum(IGP metrics)

What about more complex requirements?

- Examples
  - A path where all links have at least 10 Mbps of available bandwidth
  - A path that has the smallest propagation delay and does not pass through the Atlantic Ocean
  - A path that always passes only through specific routers
  - A path that is composed of links having at least 100 Mbps of available bandwidth and does not use satellite links

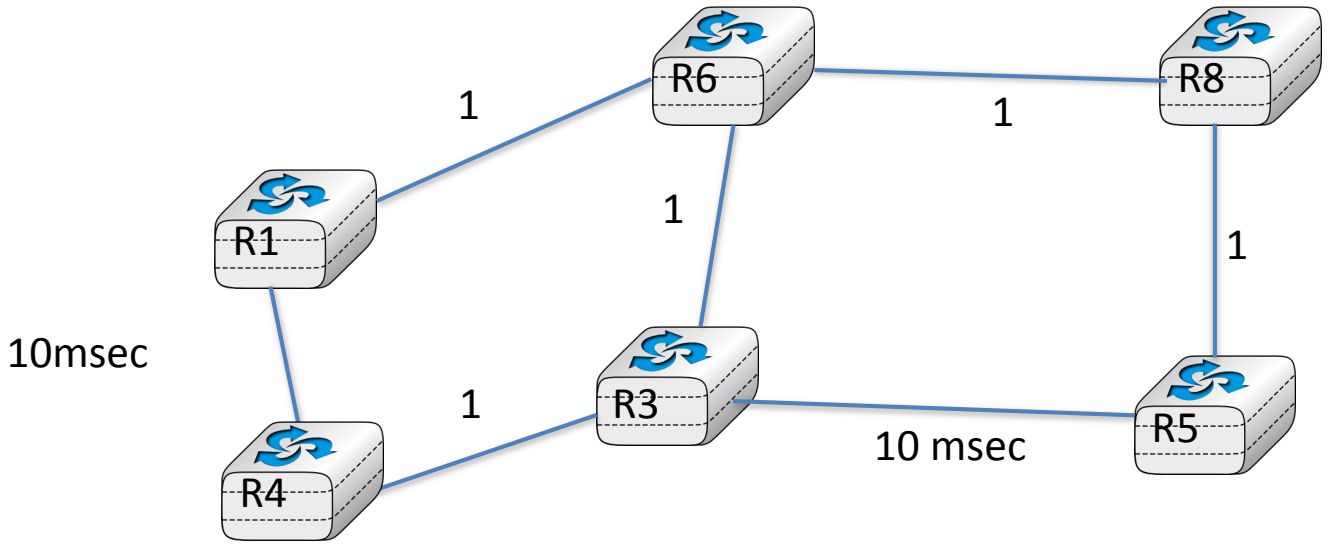
The previous examples are examples of concave constraints

- To solve them, simply
  - remove from the network map all links that do not satisfy the constraint
  - use Dijkstra's algorithm on the reduced map
  - example
    - find shortest 3 Mbps from R3 to R6



Creating TE LSPs

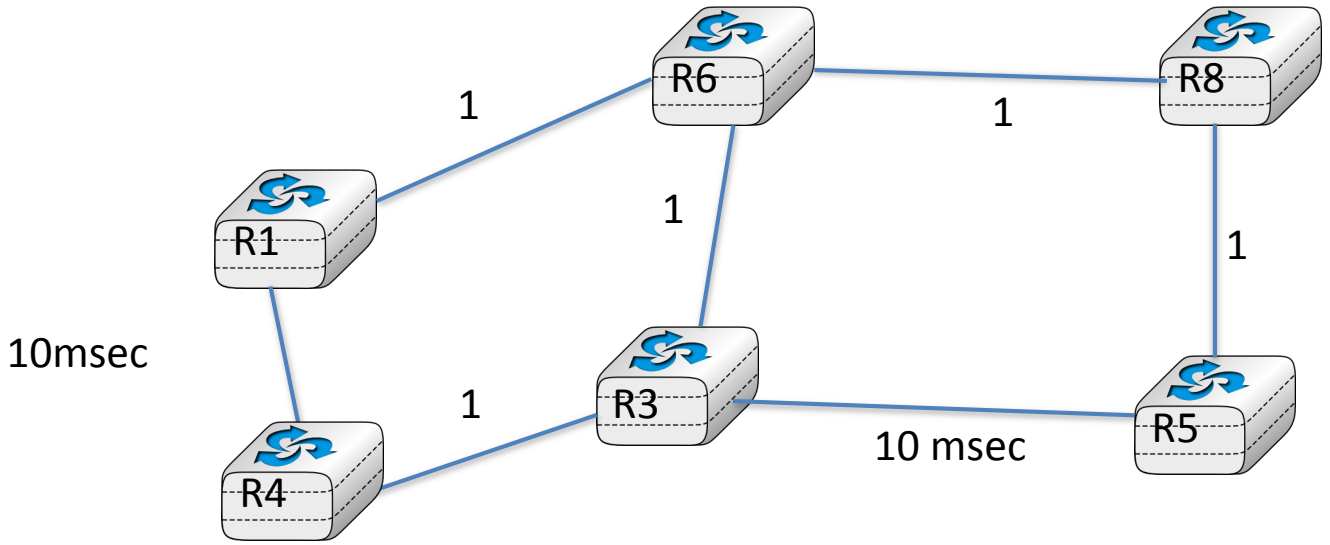
- TE LSPs
  - R4 -> R5, 600 Mbps, minimum delay
  - R1 -> R5, 600 Mbps, minimum delay
  - R4 -> R8, 300 Mbps, minimum delay



All links are 1 Gbps

Creating TE LSPs

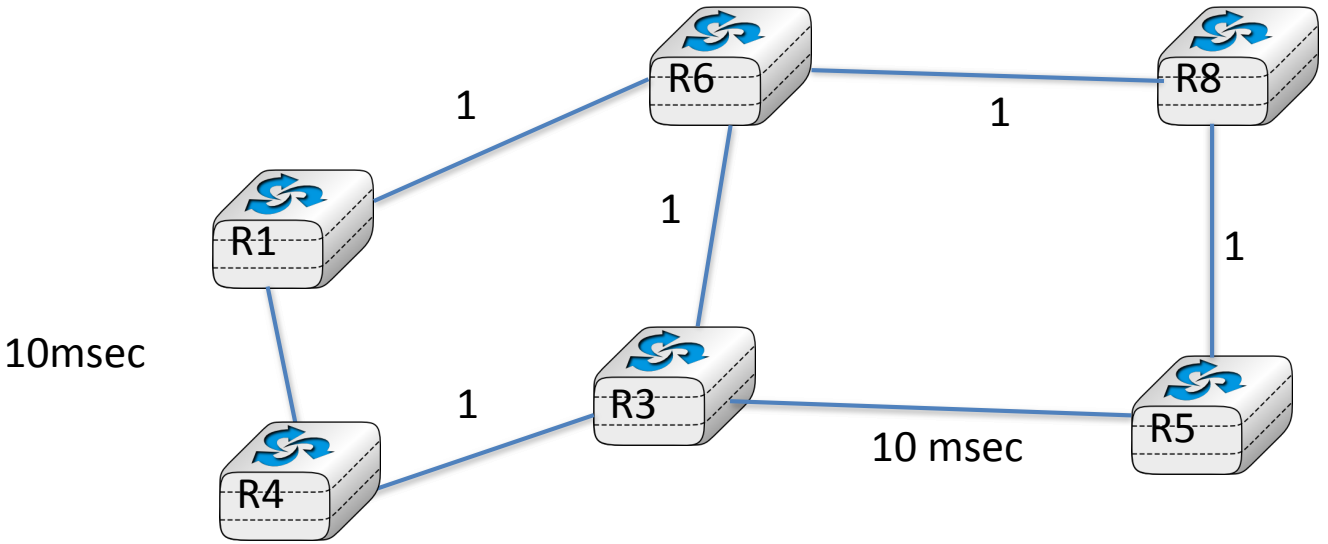
- TE LSPs
  - R4 -> R5, 600 Mbps, minimum delay
  - R4 -> R5, 600 Mbps, minimum delay, disjoint from previous one



All links are 1 Gbps

Creating TE LSPs

- TE LSPs
  - R4->R8, 600 Mbps
  - R1->R5, 600 Mbps
    before having received link state routing update



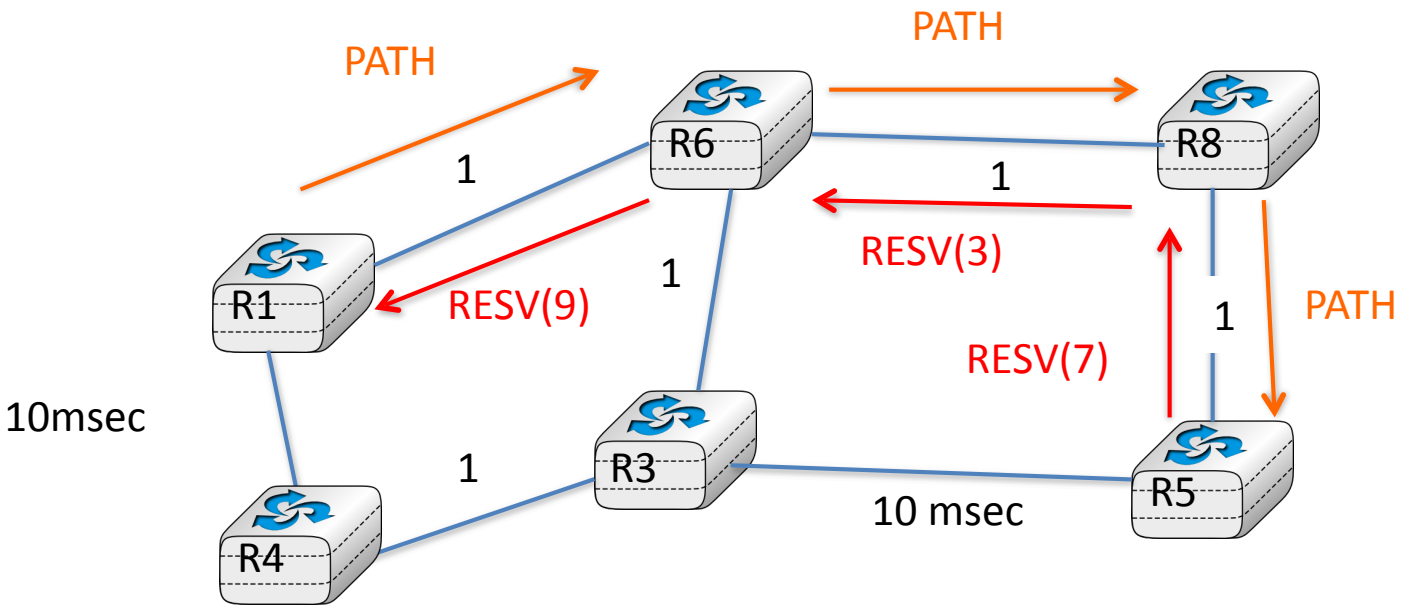
All links are 1 Gbps

Issues with TE routing

- When to transmit new link state packets ?
  - As soon as one of the TE attributes changes
    - could lead to lot of churn
  - Every *x* seconds
    - could lead to time lag between changes in TE attributes and the transmission of new link state packet
  - After a significant change in the TE attributes
    - What is a significant change?

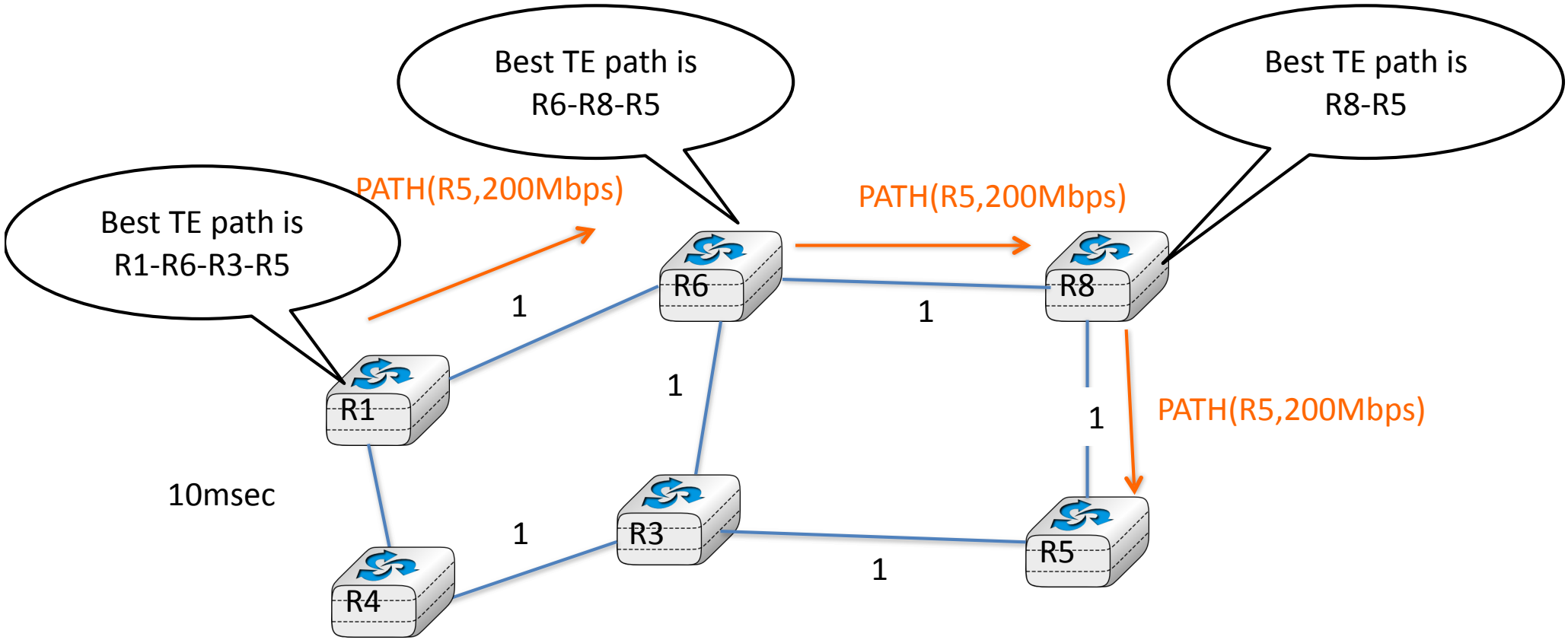
How to signal TE LSPs ?

- By using RSVP
  - PATH is used to request a label
  - RESV returns the allocated label



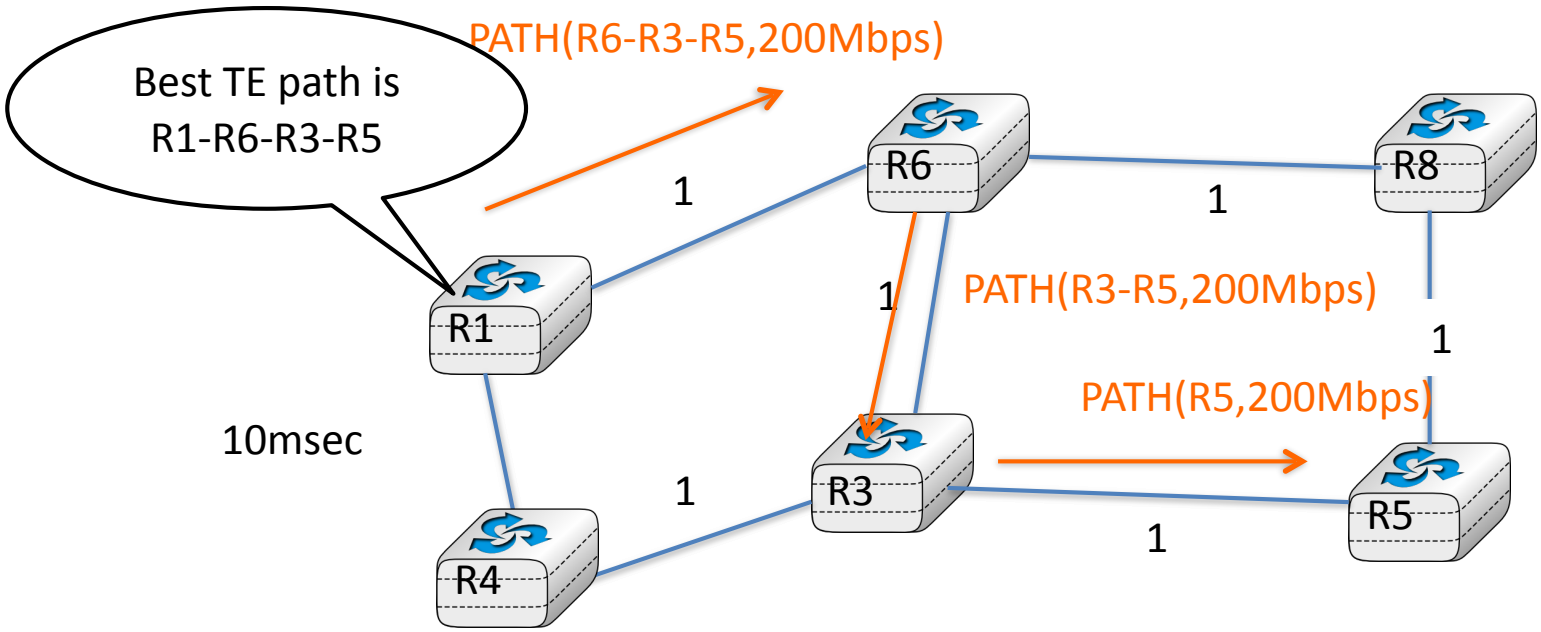
Creating TE LSPs

- How to route PATH message ?
  - Hop-by-hop routing



Creating TE LSPs

- How to route PATH message ?
  - Explicit routing



RSVP in details

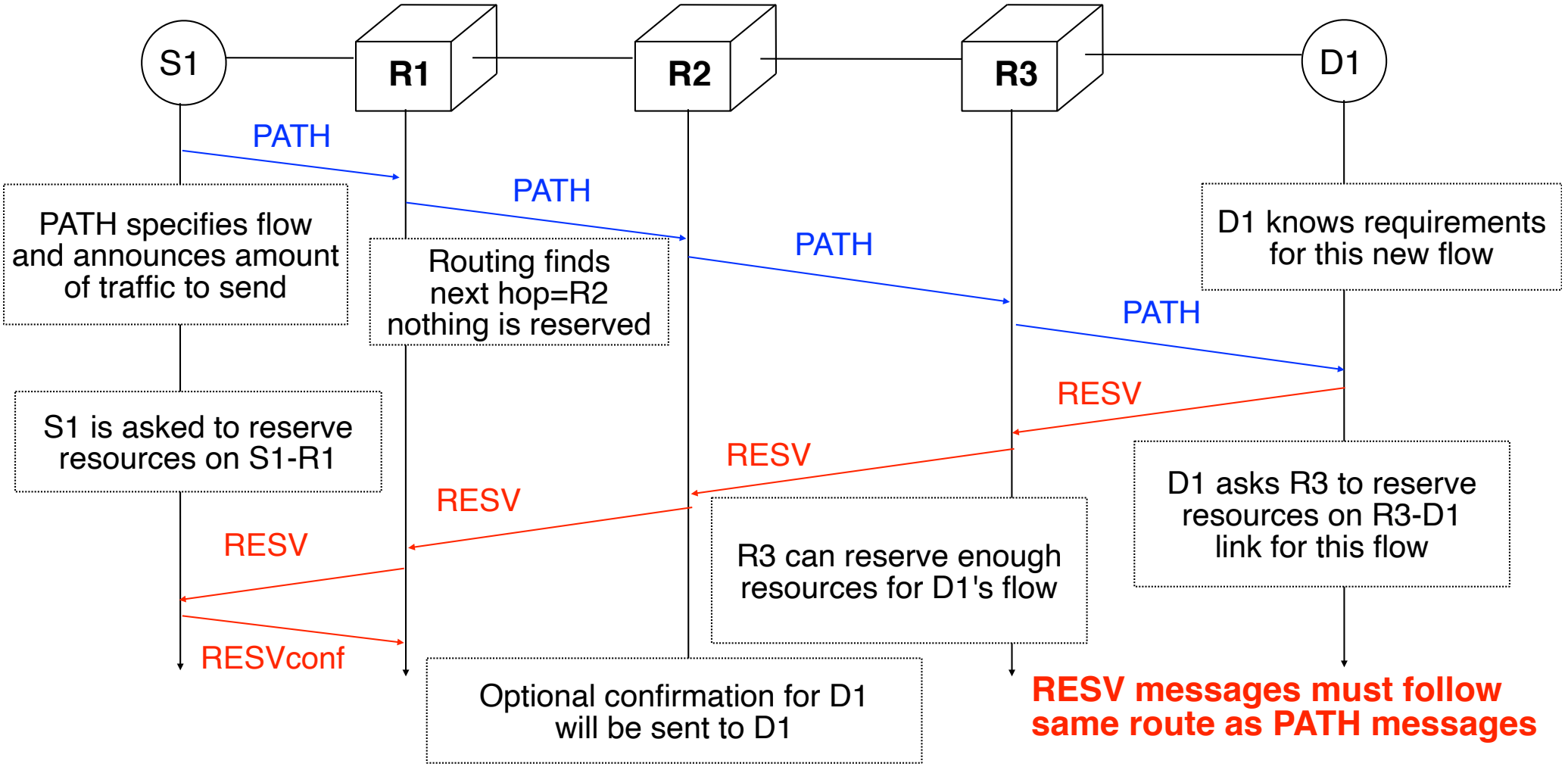
- RSVP: Resource Reservation Protocol
- Designed to support the establishment of unidirectional flows in IP networks
  - initially layer 4 flows
  - today mainly for MPLS LSPs

RSVP (2)

- Principles of operation
- Two important RSVP messages
  - PATH
    - used by sender to inform routers and receivers of the new flow and its required resources
      - no resources are reserved due to reception of PATH
  - RESV
    - used by receiver to actually reserve resources for the flow specified in the PATH message
      - resources are reserved for the IP packets sent by the sender towards the receiver along the path taken by the PATH message
- RSVP messages are sent inside IP packets

RSVP : example

- Unicast flow establishment with RSVP



RSVP messages are sent unreliably as simple datagrams

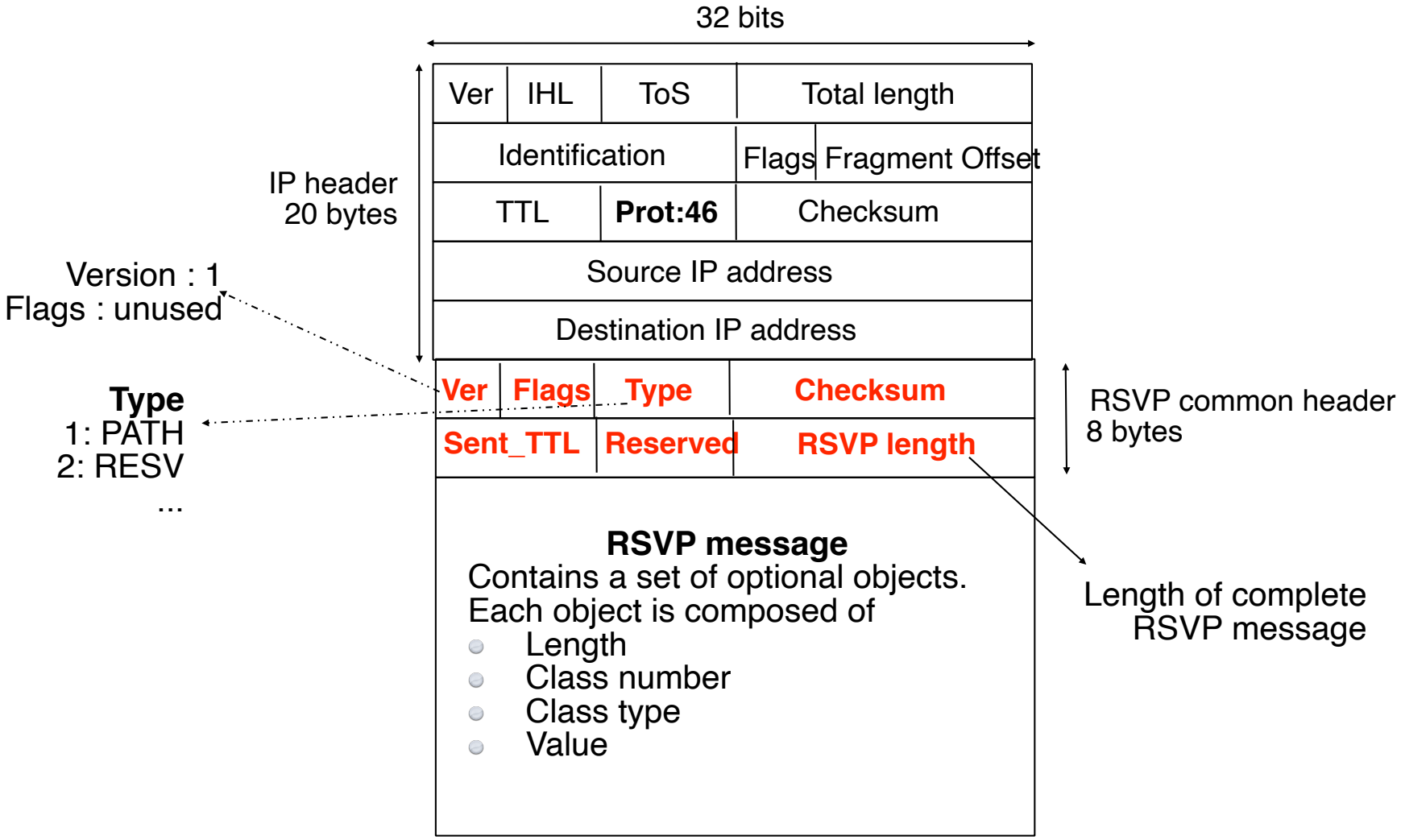
State maintenance

- RSVP routers maintain some per-flow state
- Possible solutions for state maintenance
  - Hard-state
    - traditional solution used in circuit-switched network
    - state is created at flow establishment and removed at flow tear down
    - if intermediate router crashes, state+reservation are lost
  - Soft-state (solution chosen by RSVP)
    - a timer is associated with each per-flow state
      - state is removed upon expiration of timer
    - hosts periodically retransmit PATH and RESV message
      - timer is reset upon reception of PATH/RESV message
    - if intermediate router crashes, state automatically reset
    - if route changes, new reservations/states are established on new path and all reservations/states expires

RSVP: detailed example

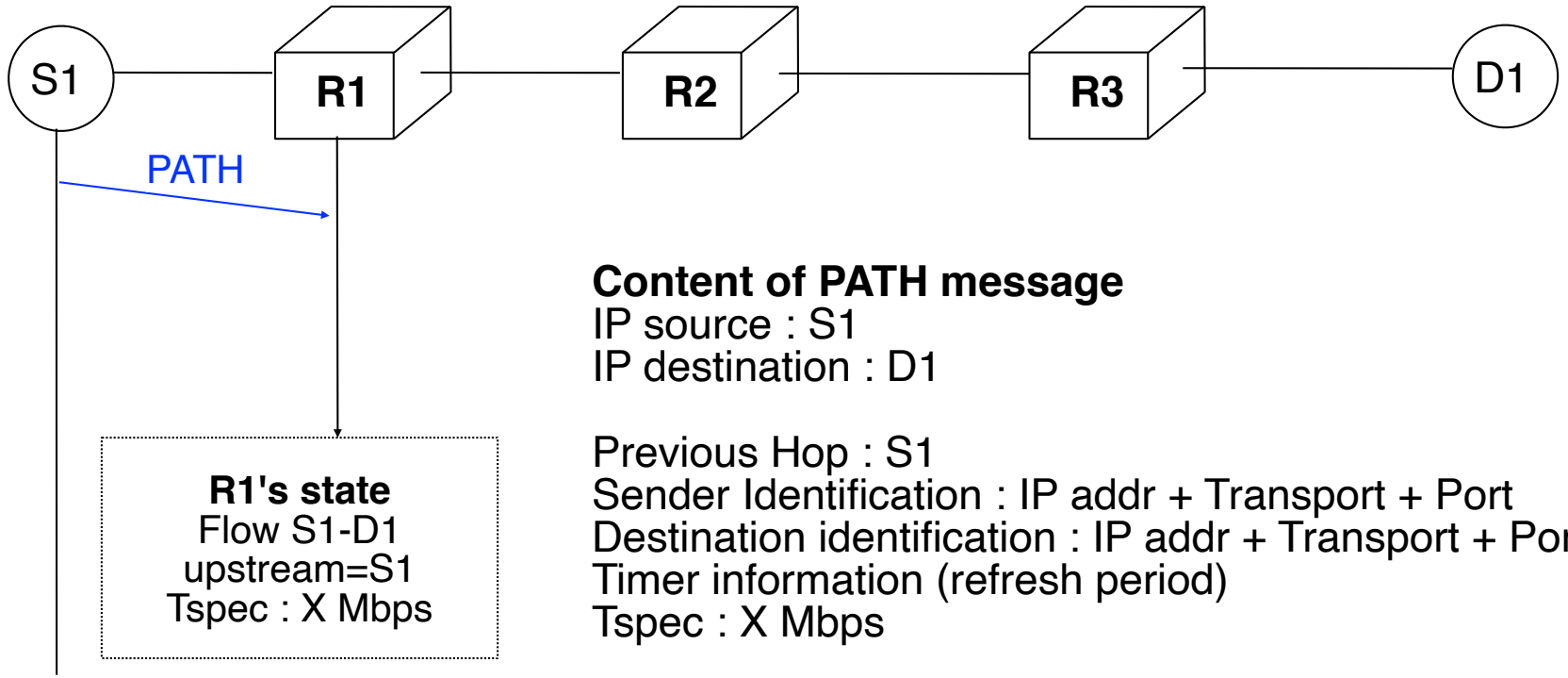
- Issues to consider
  - How to encapsulate RSVP messages in IP packets?
  - How to ensure that the reservation messages follow the same route as the PATH messages
    - cannot simply rely on IP for this!
  - What kind of information is stored in the intermediate routers?

RSVP: packet format



RSVP : detailed example (1)

- What happens inside the routers ?
  - S1 announces a flow towards D1

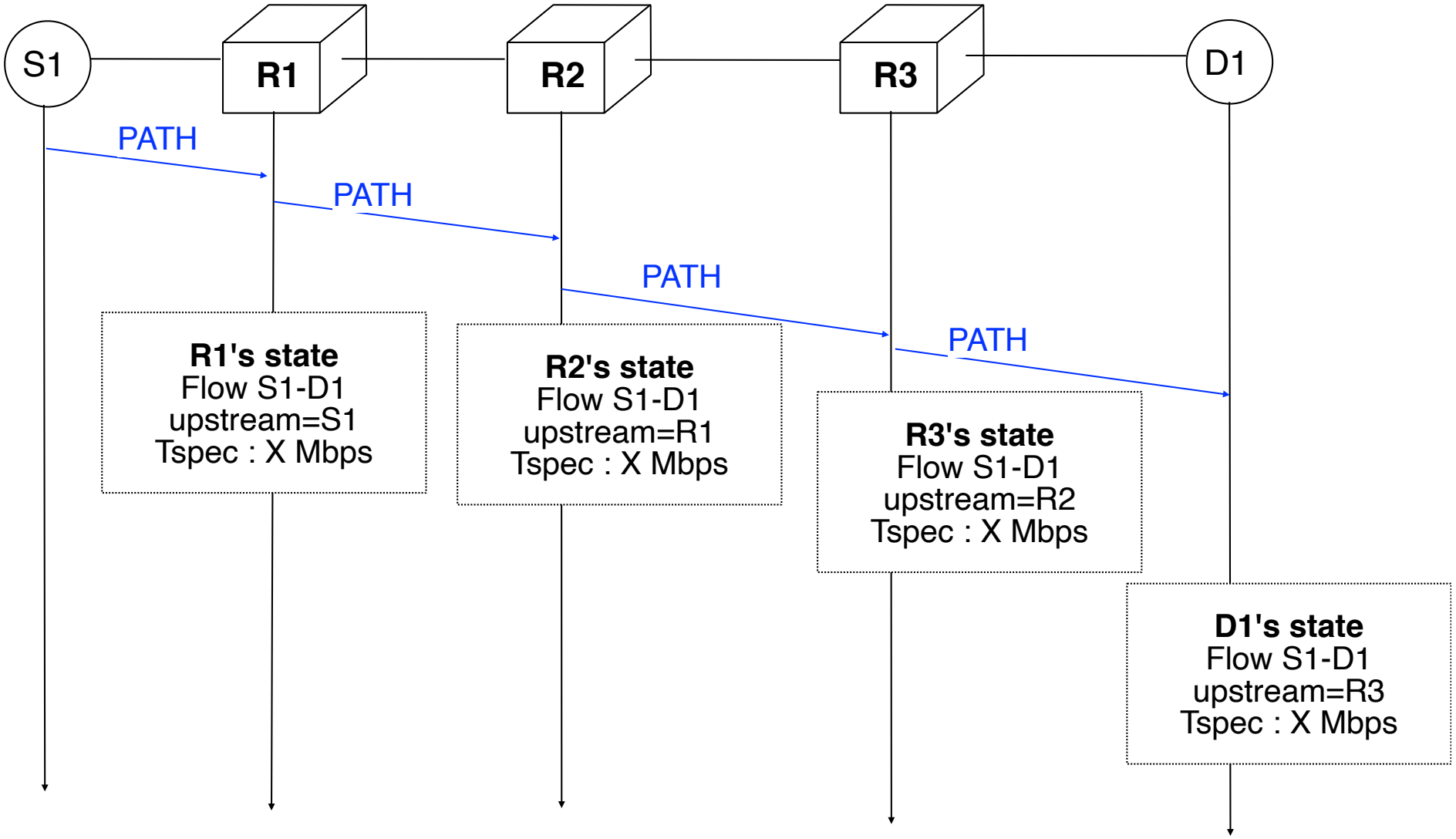


Content of PATH message
IP source : S1
IP destination : D1

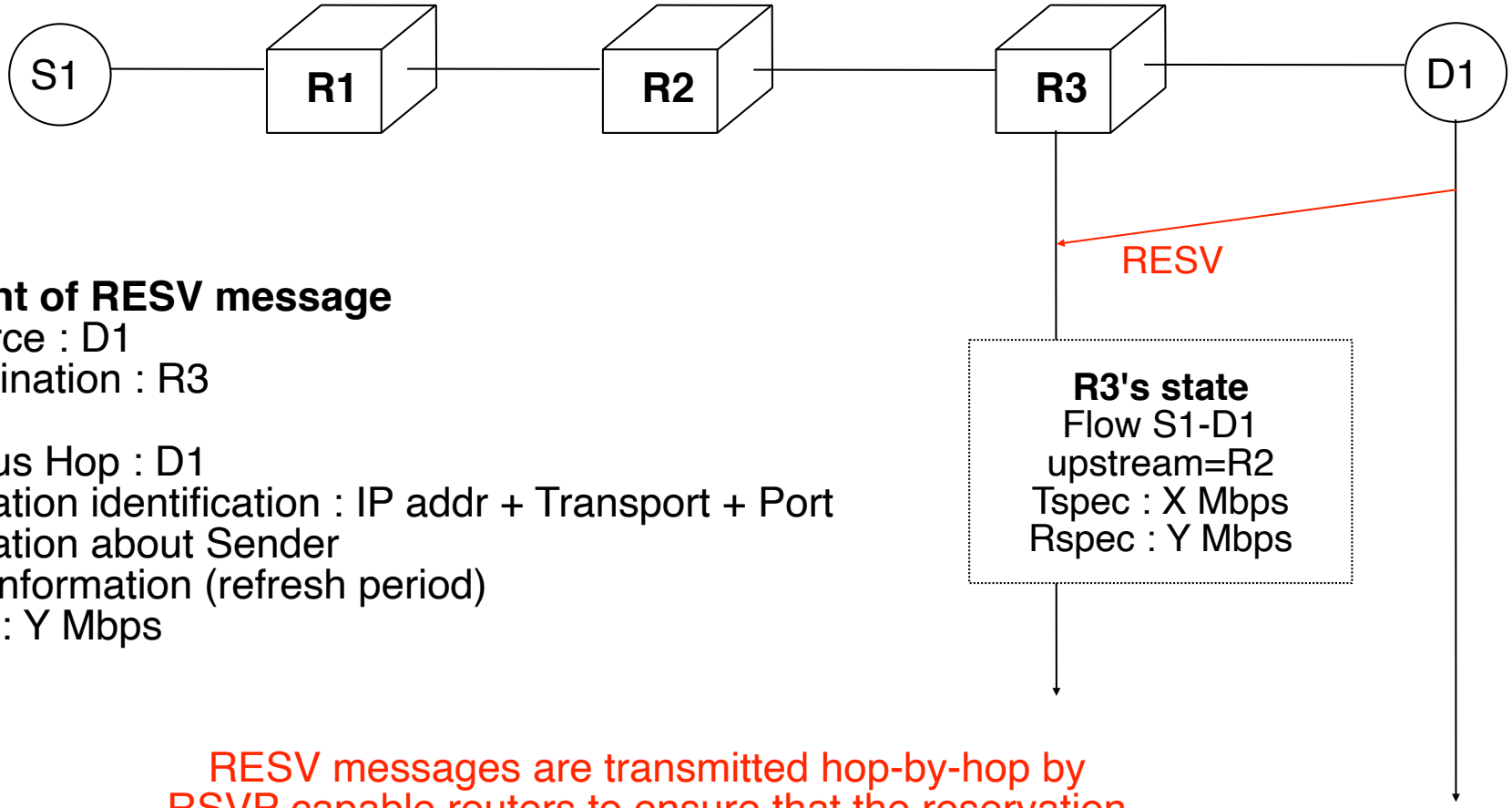
Previous Hop : S1
Sender Identification : IP addr + Transport + Port
Destination identification : IP addr + Transport + Port
Timer information (refresh period)
Tspec : X Mbps

Behaviour of R1
Content of PATH message inserted in R1's
RSVP message updated and sent downstream
PHOP replaced by R1

RSVP : detailed example (2)



RSVP : detailed example (3)



Content of RESV message
IP source : D1
IP destination : R3

Previous Hop : D1
Destination identification : IP addr + Transport + Port
Information about Sender
Timer information (refresh period)
Rspec : Y Mbps

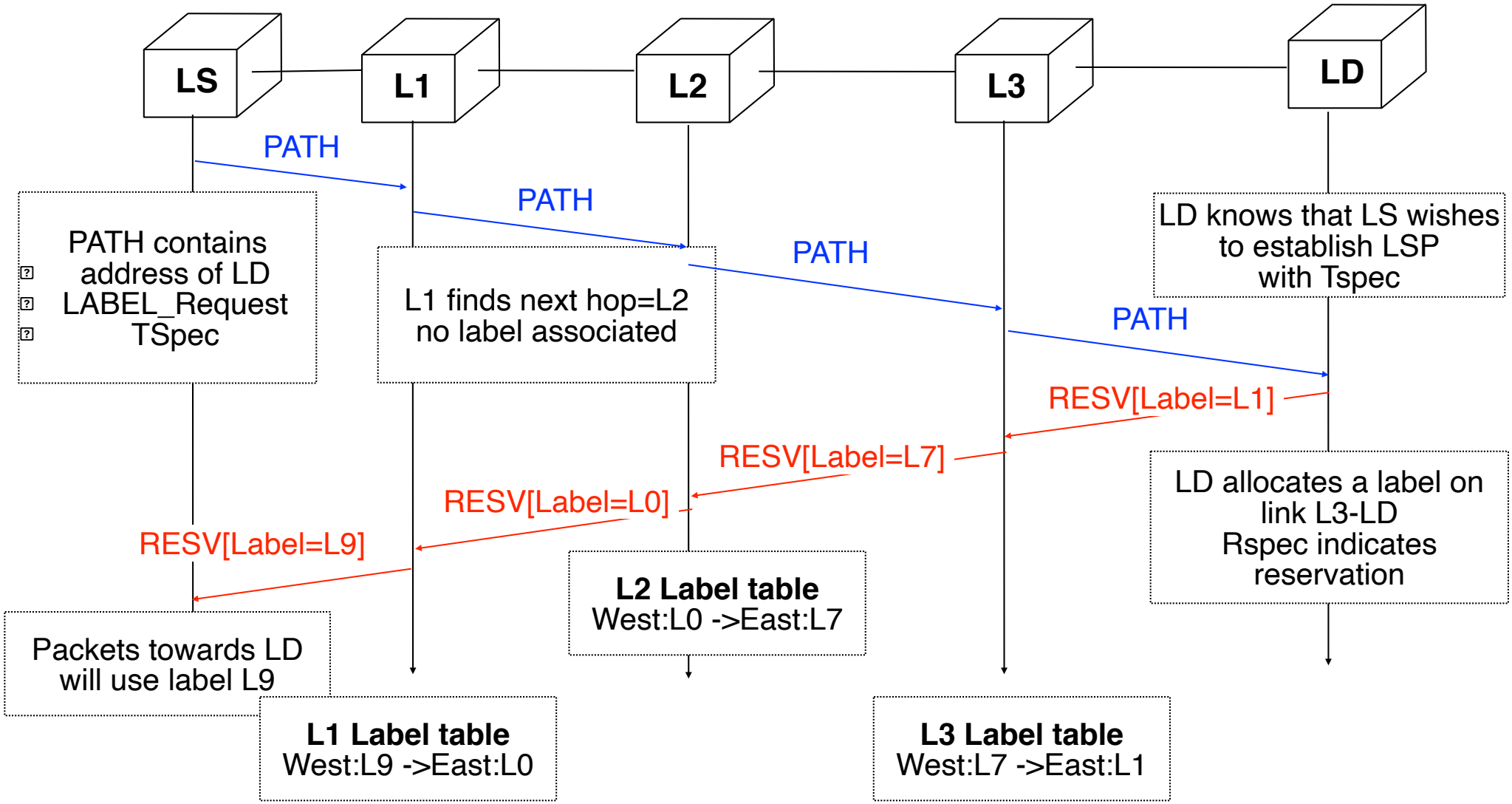
RESV messages are transmitted hop-by-hop by RSVP capable routers to ensure that the reservation follows the same route as the corresponding PATH

Using RSVP to distribute MPLS labels

- Principle
  - RSVP supports downstream on-demand label allocation
    - RSVP extension for MPLS called RSVP-TE
  - Ingress LSR sends PATH message towards egress LSR
    - PATH message includes Label Request Object
  - Egress LSR sends label back in RESV message
    - RESV propagates the labels hop-by-hop

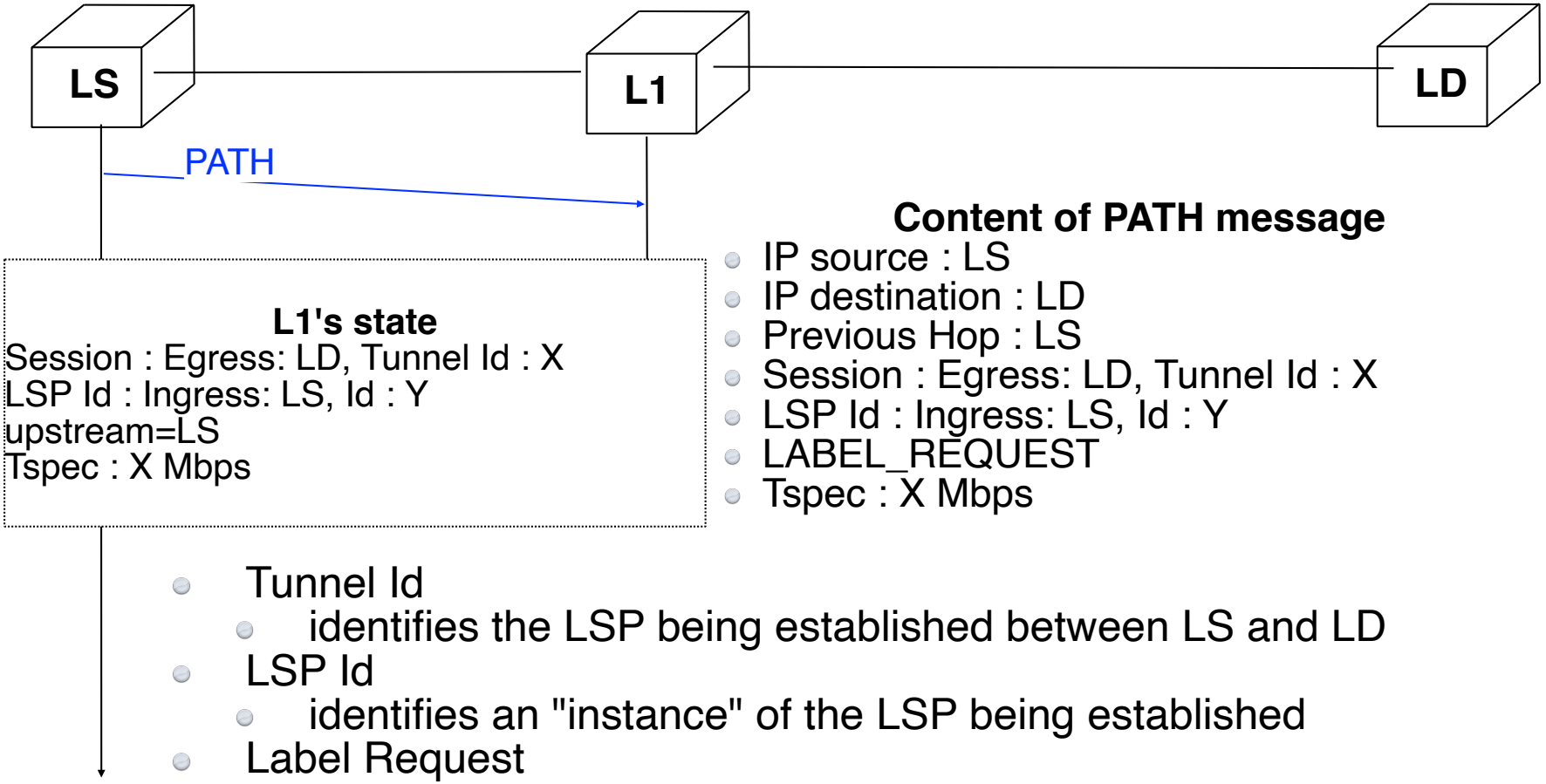
Using RSVP to distribute MPLS labels (2)

- LSP establishment with RSVP-TE



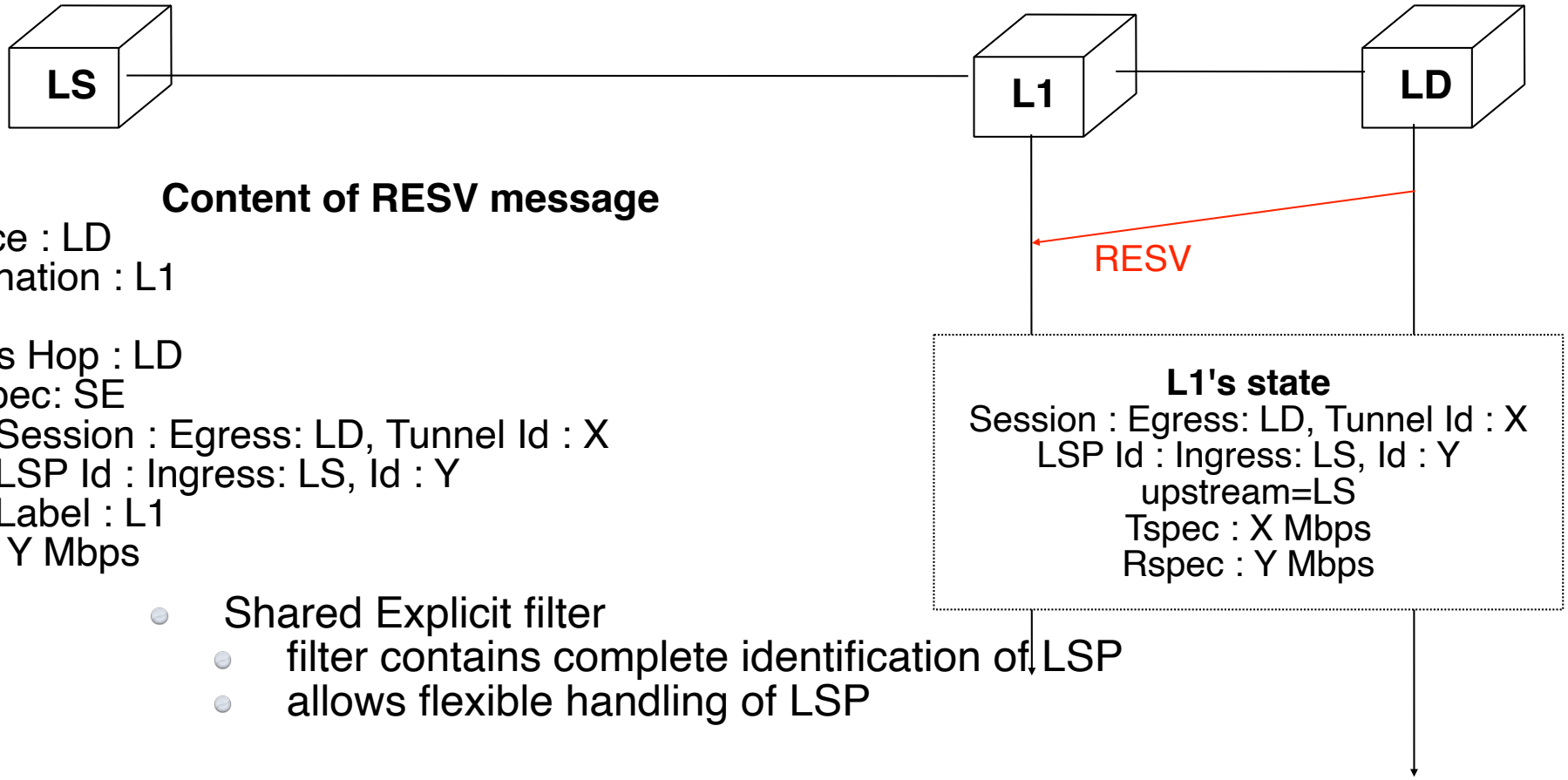
RSVP-TE : detailed example

- What happens inside the LSR ?



RSVP-TE : detailed example (2)

- Content of the RESV message



Content of RESV message

IP source : LD
IP destination : L1

Previous Hop : LD
Filter Spec: SE
Session : Egress: LD, Tunnel Id : X
LSP Id : Ingress: LS, Id : Y
Label : L1
Rspec : Y Mbps

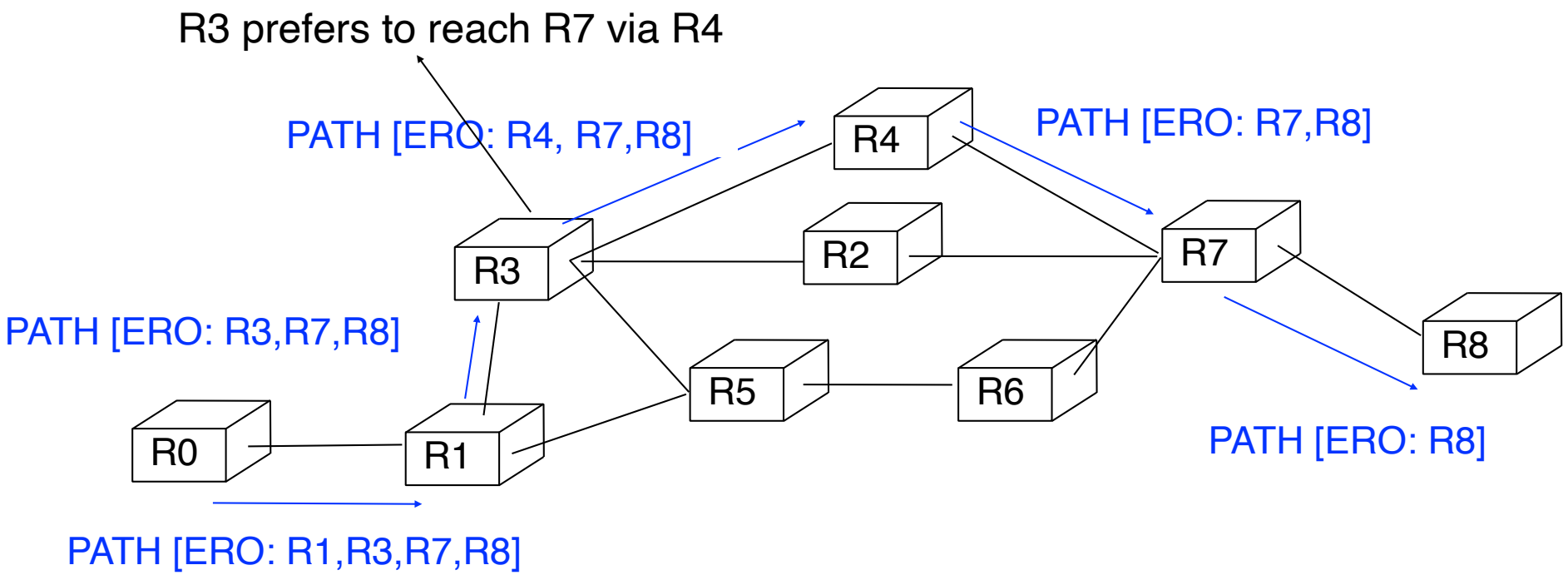
- Shared Explicit filter
  - filter contains complete identification of LSP
  - allows flexible handling of LSP

Identification of LSP : [Ingress, Egress, Tunnel ID, LSP Id]

How to establish/signal LSP along non-shortest path?

- Ingress LSR may specify the route to be followed by an LSP being established
- Route specification is composed of a list of
  - IP addresses
  - Subnet prefixes
  - Autonomous System numbers
- Two types of route specifications:
  - *Strict* route
    - the LSP must pass through each LSR specified by ingress LSR
  - *Loose* route
    - the LSP can pass through non-specified LSR between two specified LSRs

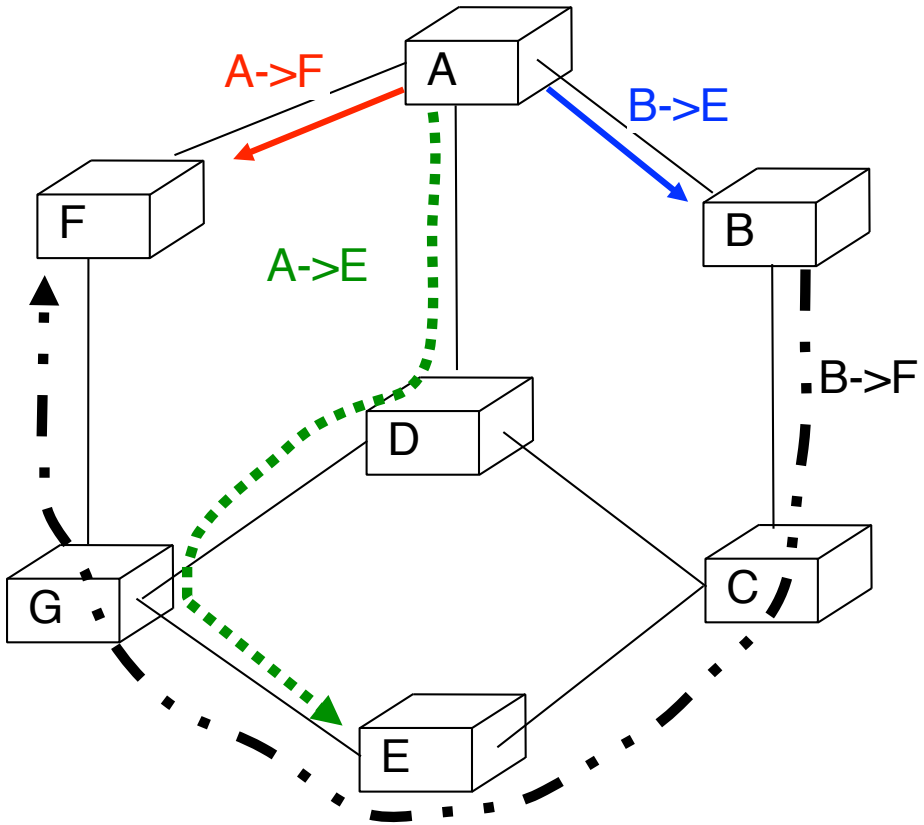
RSVP-TE : Explicit Routes (2)



Traffic engineering with MPLS

A's routing table
F: d=1 : West
D: d=1 : South
B: d=1 : East
G: d=2 : West
C: d=2 : South
E: d=3 : South

A's mapping table
Destination Label
B LB
E LE
F LF

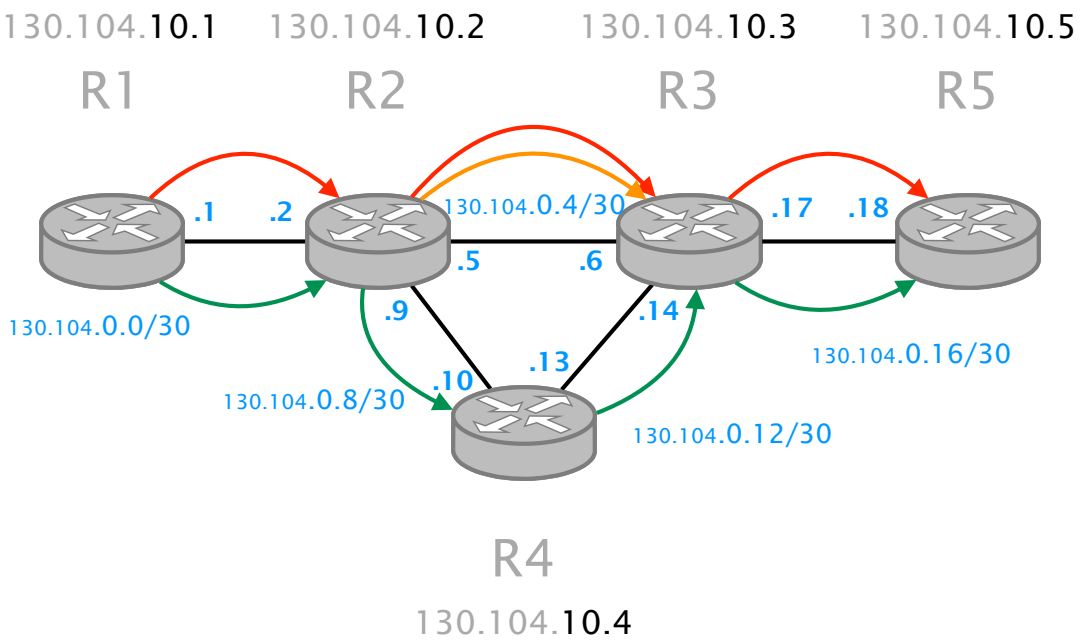


B's mapping table
Destination Label
F LF

Traffic matrix
A->B : 0.6
A->F : 0.6
B->F : 0.6
A->E : 0.6

B's routing table
A: d=1 : North
C: d=1 : South
D: d=2 : South
E: d=2 : South
F: d=2 : North
G: d=3 : South

Let's play in the lab



LSP	request order
80 Mbps	1
20 Mbps	2
20 Mbps	3

Advanced Topics in Communication Networks

Internet Routing and Forwarding

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

13 Oct 2020