Advanced Topics in Communication Networks

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich

Tue 20 Sep 2022

What network-related problems did *you* experience recently?

In this lecture, you'll learn…

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

of network infrastructures

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

of network infrastructures

Why should ***you*** care?

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

of network infrastructures

Why should ***you*** care? as a user?

network operator?

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

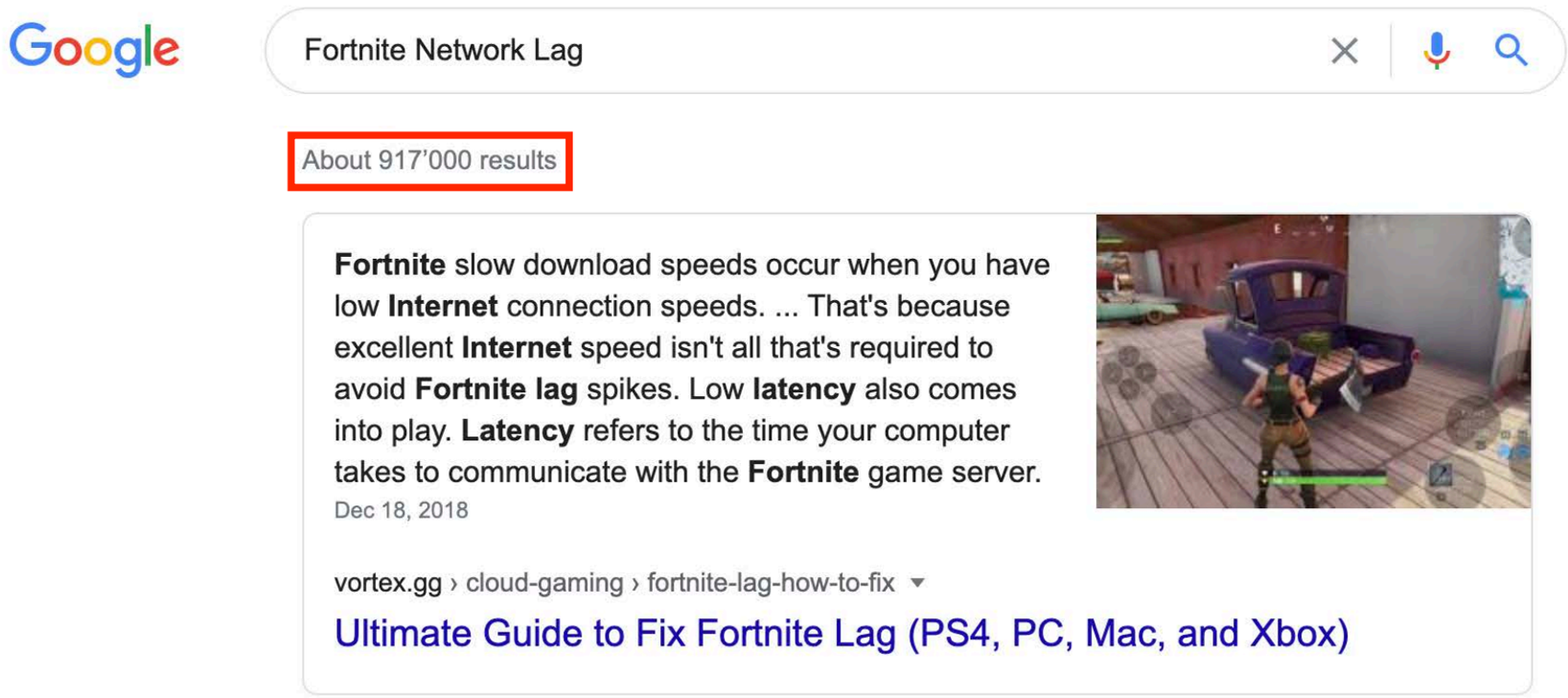
of network infrastructures

Why should ***you*** care? as a **user?**

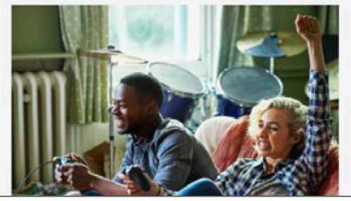
network operator?

Fortnite, Epic Games

Many gamers care about low latency!



The 3 L's of Gaming – Location, Location & .... Latency !!!



Brenden Rawle
Director of Interconnection in EMEA

Martin Atkinson
Senior Manager of Peering and Interconnection EMEA, Equinix

October 1, 2019 • 7 min read

Share:

44% of UK Online Multiplayer Gamers Still Complain About Lag

Tuesday, June 23rd, 2020 (2:21 pm) - Score 4,169

Email | Link News 52 Comments

A new survey of 1,000 UK video gamers, l fibre found that d "*the* uriating 7% also slow dents are 5 hours 3 hours lf said the t have Meanwhile rs.

Players Report Huge Lag Spikes & Server Issues in Warzone Season 5

WARZONE

By **Django Zimmatore**
Published: 19 August 2021, 03:35
Updated: 19 August 2021, 03:35

Warzone Season 5 is here but it's been far from smooth – many players are suffering from terrible lag spikes and server issues after the update.

The new Season of Warzone has introduced a whole **load of new content to the game**. This includes new weapons, operators, and locations in Verdansk.

However, many players think that the new update has introduced some problems too. Now, many are suffering from terrible lag spikes and server issues in Warzone.

But first, find out why players think Warzone Season 5 is the worst update ever.

How to reduce lag in PC games

By NICK GREENE Last updated 26 Mar 2020

Source: Pixab

Wh
yo
fru
Int
my

Actiontec PRODUCTS SERVICES RESOURCES LEARN SUPPORT

How to Reduce Latency or Lag in Gaming

s and Gateways G.hn

o gaming. Low latency means there is a e chance for a connection loss or delay. ean instant death. A wired connection is iminates the possibility of lag.

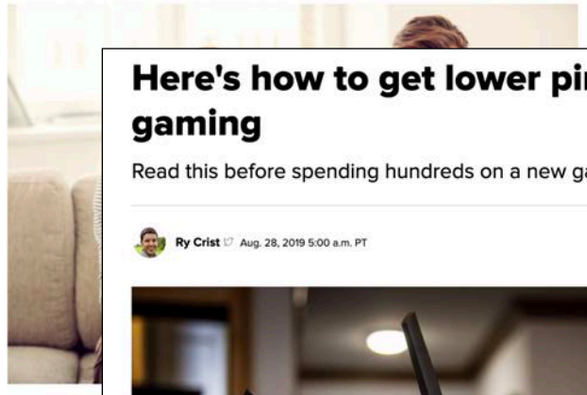
g. Then we'll look at some opportunities vork.

—

7 Ways to Reduce Latency in Online Gaming

Sarah Pike
January 4, 2016

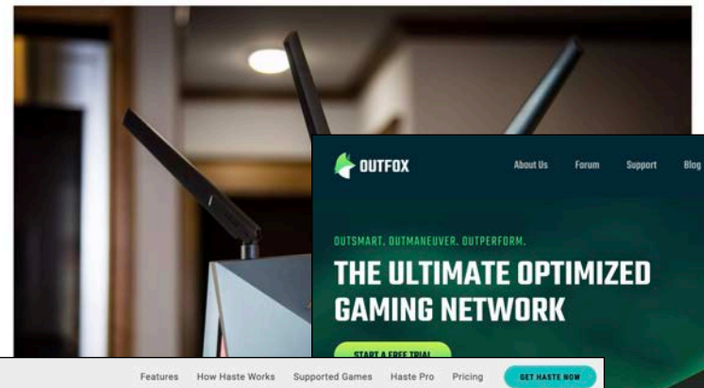
0
Comments



Here's how to get lower ping for online gaming

Read this before spending hundreds on a new gaming router.

Ry Crist Aug. 28, 2019 5:00 a.m. PT



OUTFOX About Us Forum Support Blog Download LOG IN START A FREE TRIAL

OUTSMART, OUTMANEUVER. OUTPERFORM.

THE ULTIMATE OPTIMIZED GAMING NETWORK

START A FREE TRIAL

MPROVES GAME CONNECTIONS

HASTE Features How Haste Works Supported Games Haste Pro Pricing GET HASTE NOW

RECLAIM YOUR CONNECTION

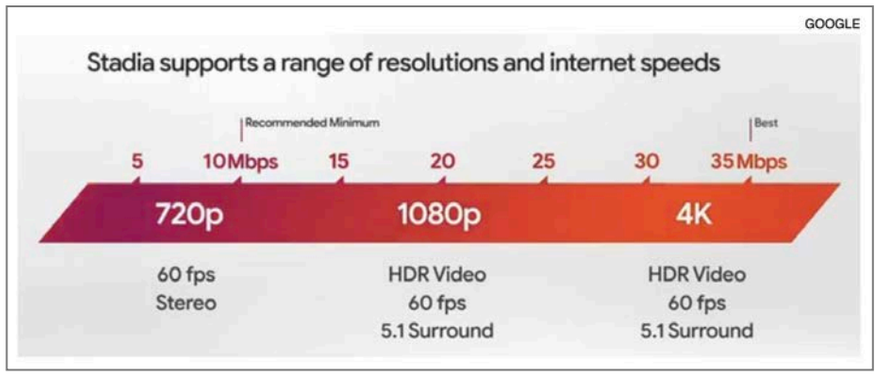
DEFEAT PING AND STABILIZE YOUR CONNECTION

GET HASTE NOW

No Credit Card Required for Haste Free

The problem of poor network performance will intensify as video games go streaming-based

Google Stadia



How to improve your internet connection for Google Stadia

Kyle Bradshaw - Nov. 19th 2019 12:30 pm PT @SkylledDev

2 Comments | Facebook | Twitter | Pinterest | LinkedIn | Reddit

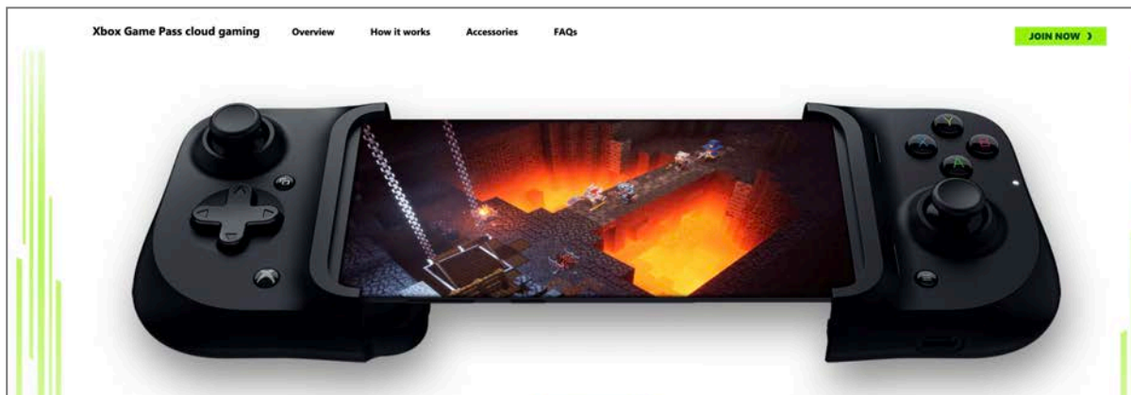
Google Stadia, as a game streaming service that can run at up to 4k and 60 frames per second, is very demanding on your internet connection. So we've collected a few ways you can improve your internet connection to Google Stadia to be both better and faster and have less lag.

Before we can dive into the different methods you can use to improve your connection, let's take a second to discuss the two main attributes of your network that can affect Google Stadia or introduce lag.

The first is your download speed, or how long it takes to download files. Your download speed

Steam Remote Play

Xbox Game Pass Cloud Gaming



PREMIUM

PLAYSTATION PLUS ON PC

PlayStation Plus Premium subscribers can stream and play PS4 games* from the Games Catalog and select titles from the Classics Catalog on any compatible PC, via the free app.

*Certain PS4 games in Game Catalog may not be available to stream.

You'll need:

- A PlayStation Network account with registered payment details
- A DUALSHOCK 4 wireless controller or other compatible controller
- An active PlayStation Plus Premium subscription
- An active broadband connection (we recommend 5 Mbps or higher)
- To be aged 18 or older

Download the PC app

To stream games to your PS4, PS5, or PC,
you need a broadband internet connection of at least 5Mbps.

To stream in 1080p resolution, you must [...] have
a broadband internet connection of at least 15Mbps.

Of course, audio/video-conferencing is also highly sensitive to network performance

and *many* others…

And, needless to say,
so is video streaming…

Where do these problems come from?

By default, IP networks only offer
a (possibly lousy) best effort service

By default, IP networks only offer
a (possibly lousy) best effort service

best effort service

IP routers do their best
to deliver packets to their destination
without making any promises

best effort service

IP routers do their best to deliver packets to their destination without making any promises

For many applications and users
doing "its best" is just not good enough anymore

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

of network infrastructures

Why should ***you*** care? as a user?

network operator?

IP networks carry always more critical services

IP networks carry always more critical services

Since the end of 2017,

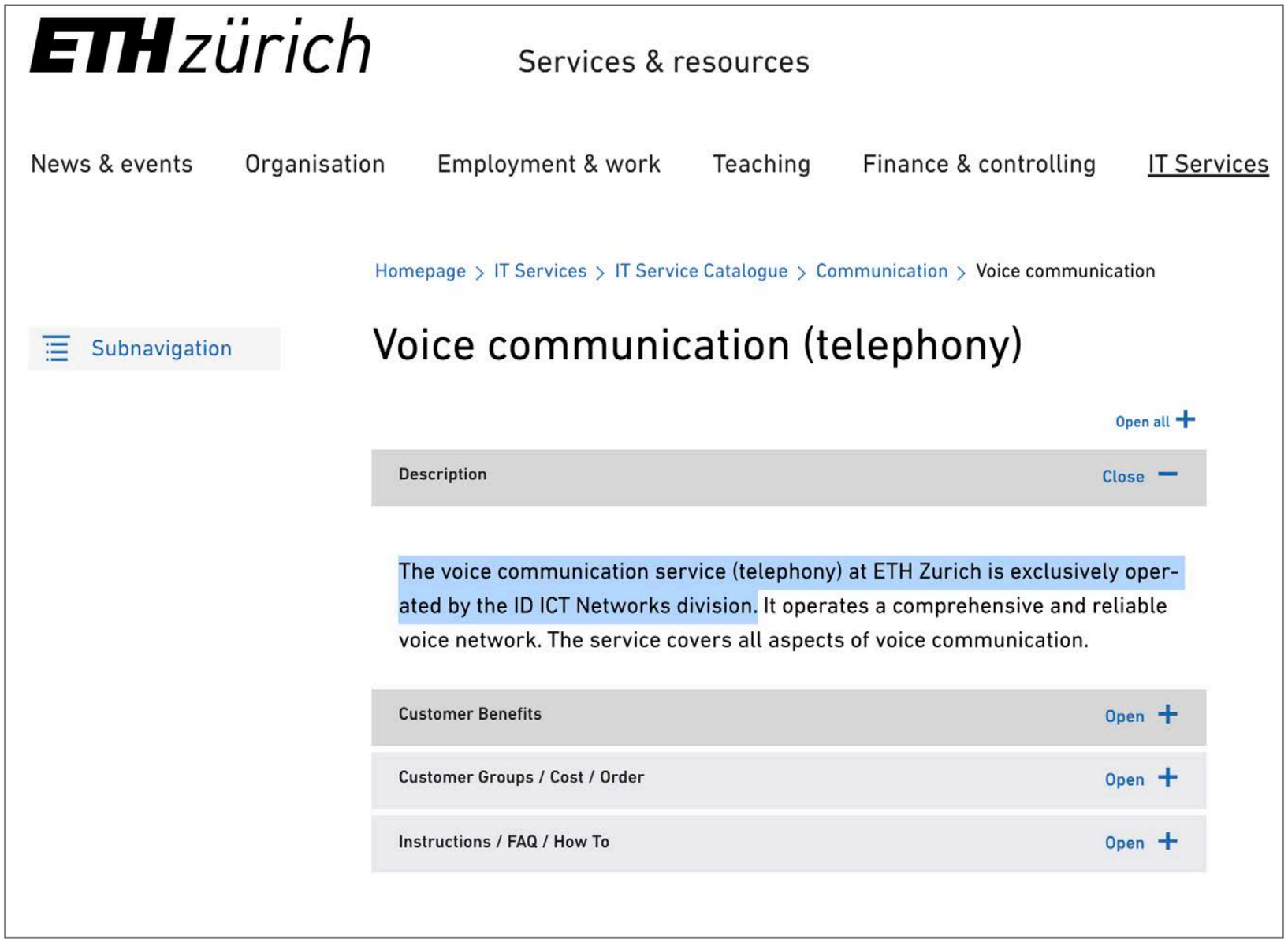
All fixed-network services

(telephony, TV, and Internet)

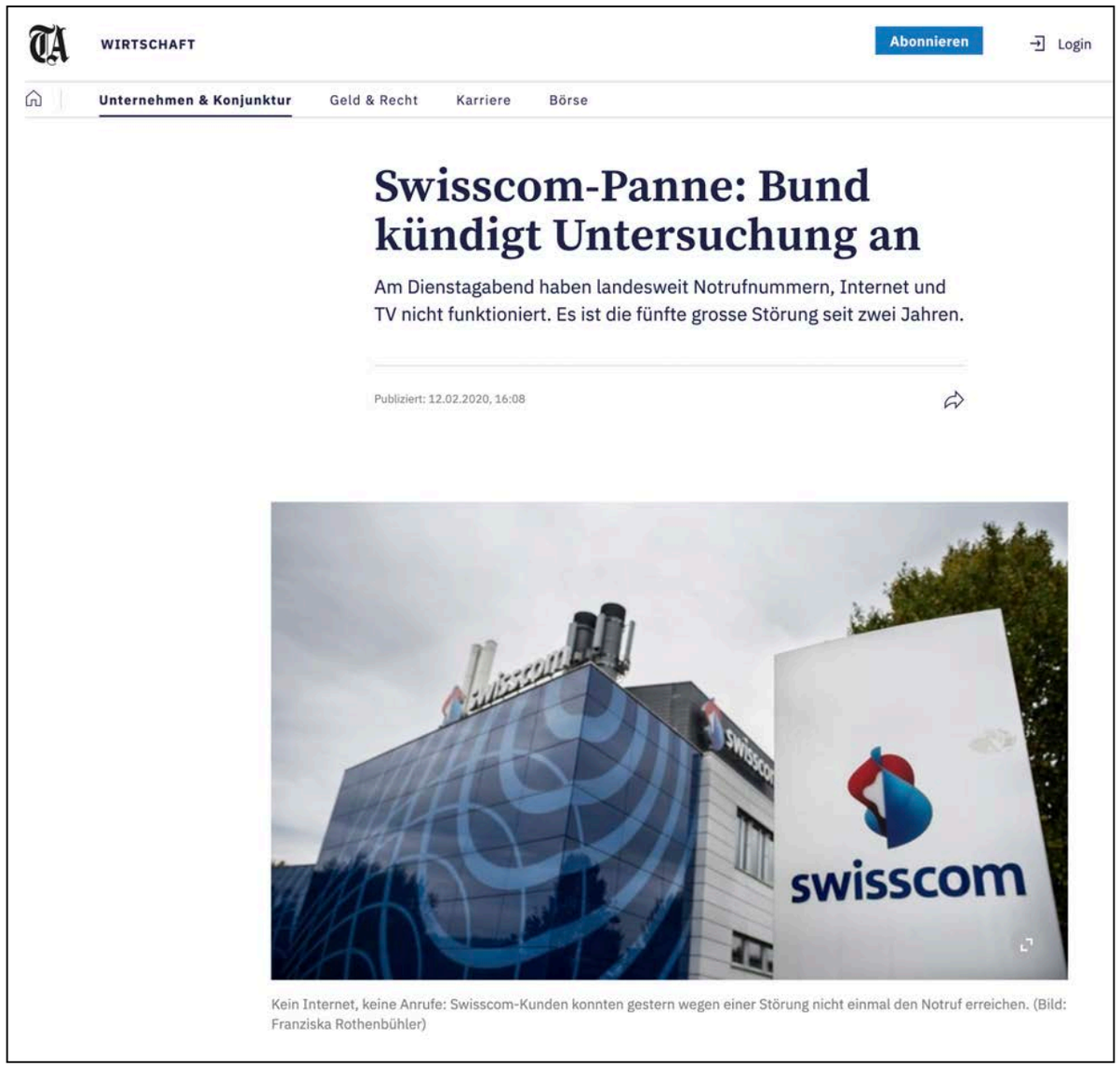
run on IP technology

Source: https://www.swisscom.ch/en/business/enterprise/themen/work-smart/all-ip.html

Same applies at ETH Zürich, whenever you pick up a phone



With great powers come great responsibilities



February 2020

Emergency numbers
(117, 112, 144, 118)
were not reachable for >1.5h
due to a major outage
in Swisscom

https://www.tagesanzeiger.ch/wirtschaft/unternehmen-und-konjunktur/bei-swisscom-laeuft-im-moment-gar-nichts/story/20660403

IP networks carry always more traffic

IP networks carry always more traffic

always faster user speeds

x

always more users

x

always more content

IP networks carry always more traffic

always faster user speeds

x

always more users

x

always more content

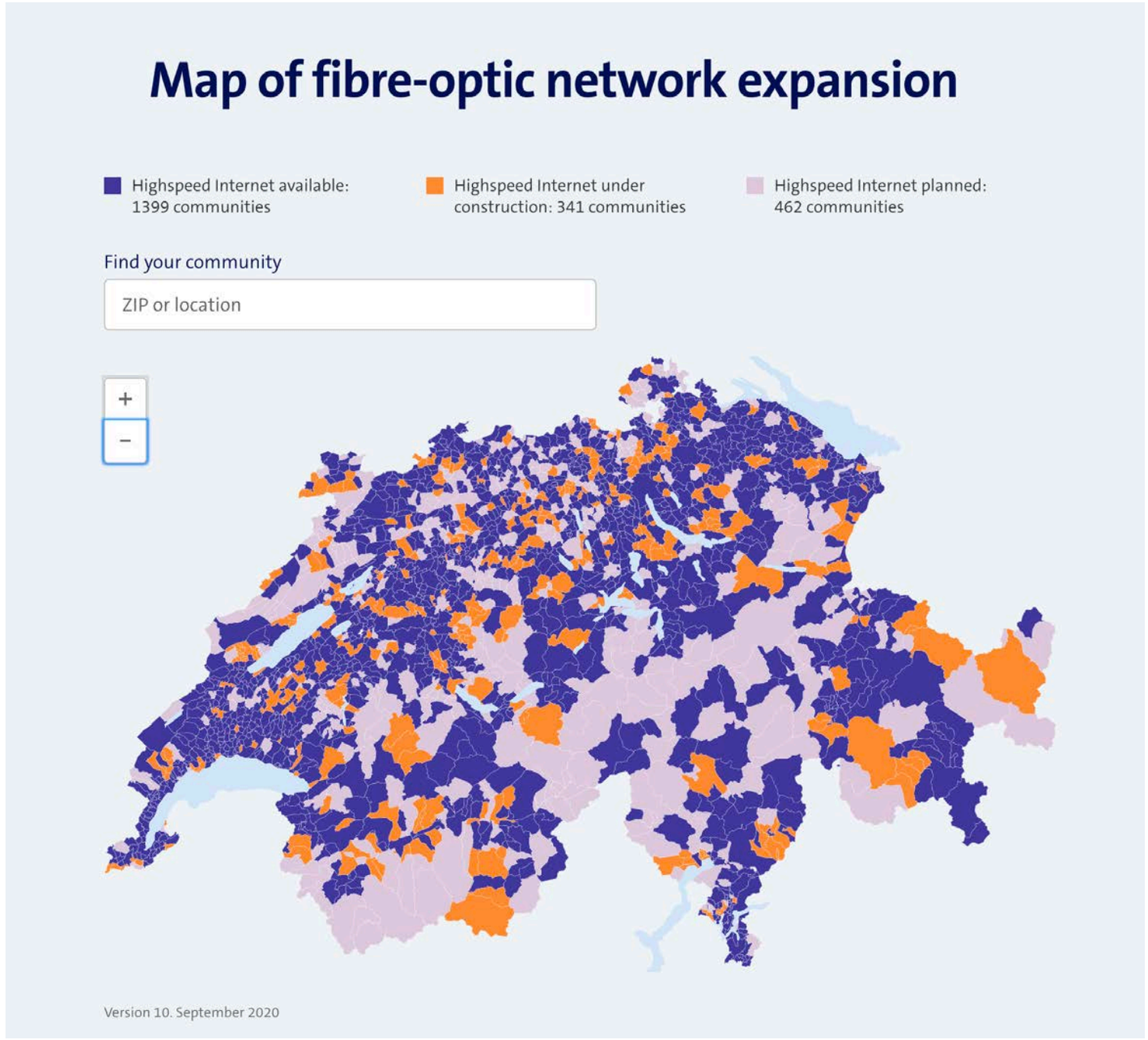
IP networks carry always more traffic

Expected growth in throughput by 2023

broadband	>x2	110 Mbps
cellular	>x3	44
Wi-Fi	>x3	92

Source: Cisco Annual Internet Report (2018–2023)

Up to 60% of Swiss homes and business will have access to Fiber connectivity by 2025



Source: Swisscom

IP networks carry always more traffic

always faster user speeds

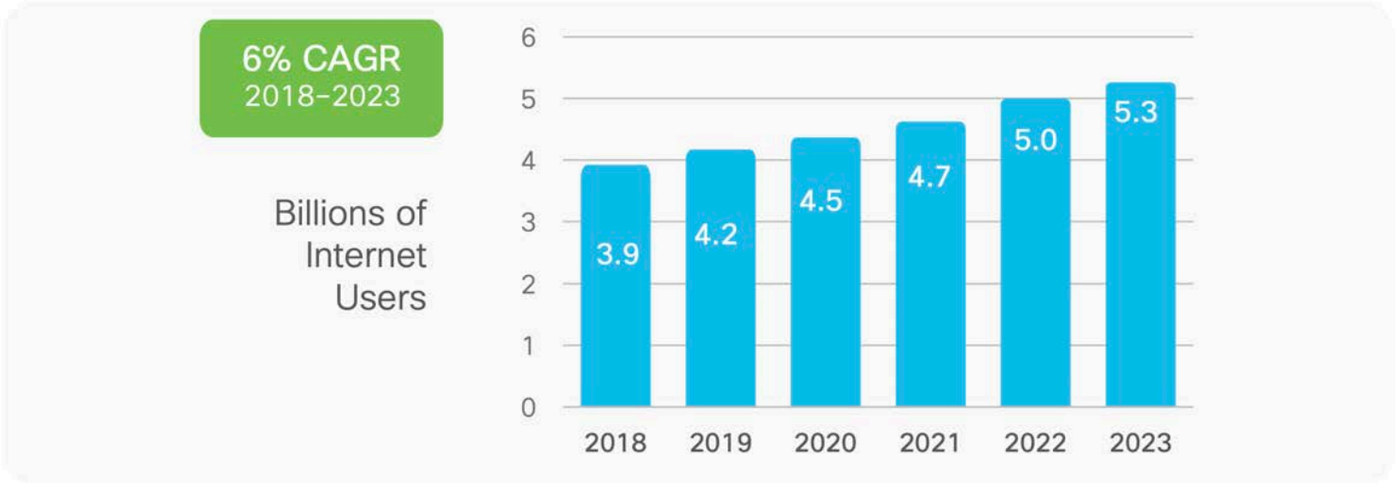
x

always more users

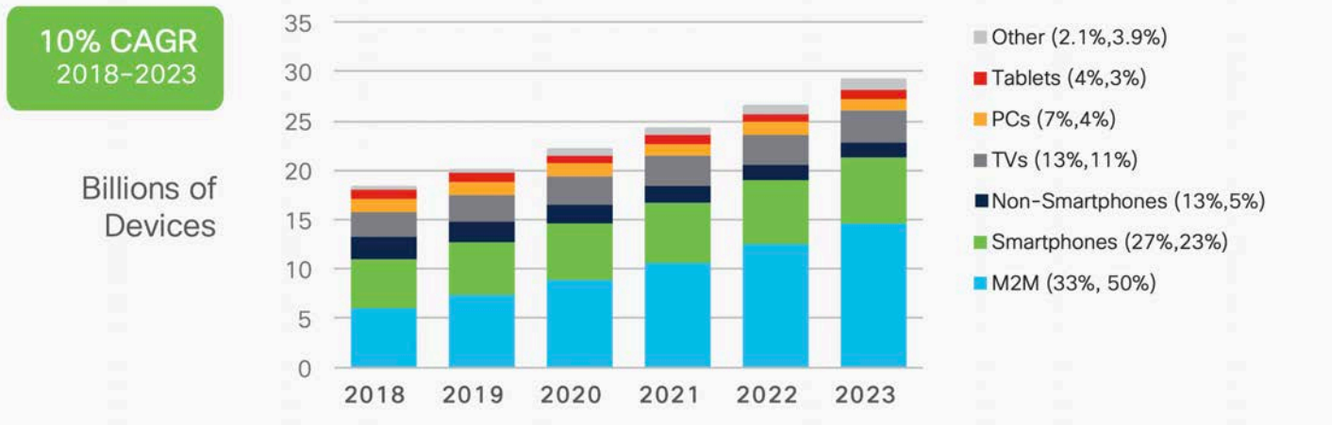
x

always more content

Internet growth is still going on strong



Source: Cisco Annual Internet Report, 2018-2023



Source: Cisco Annual Internet Report, 2018-2023

IP networks carry always more traffic

always faster user speeds

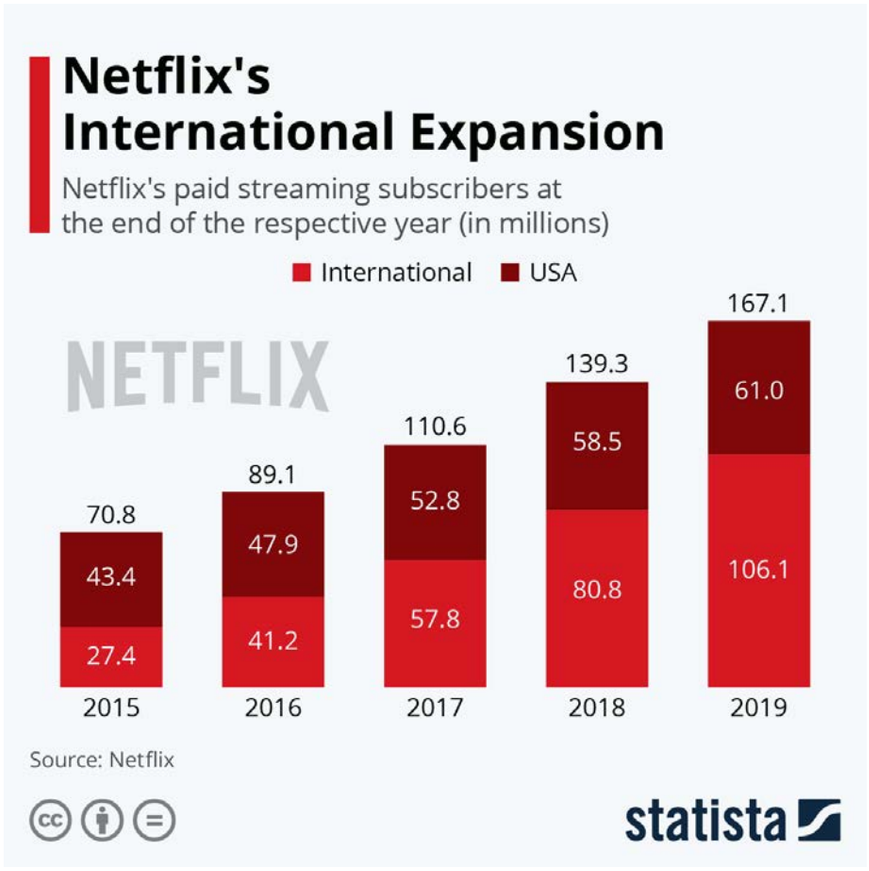
x

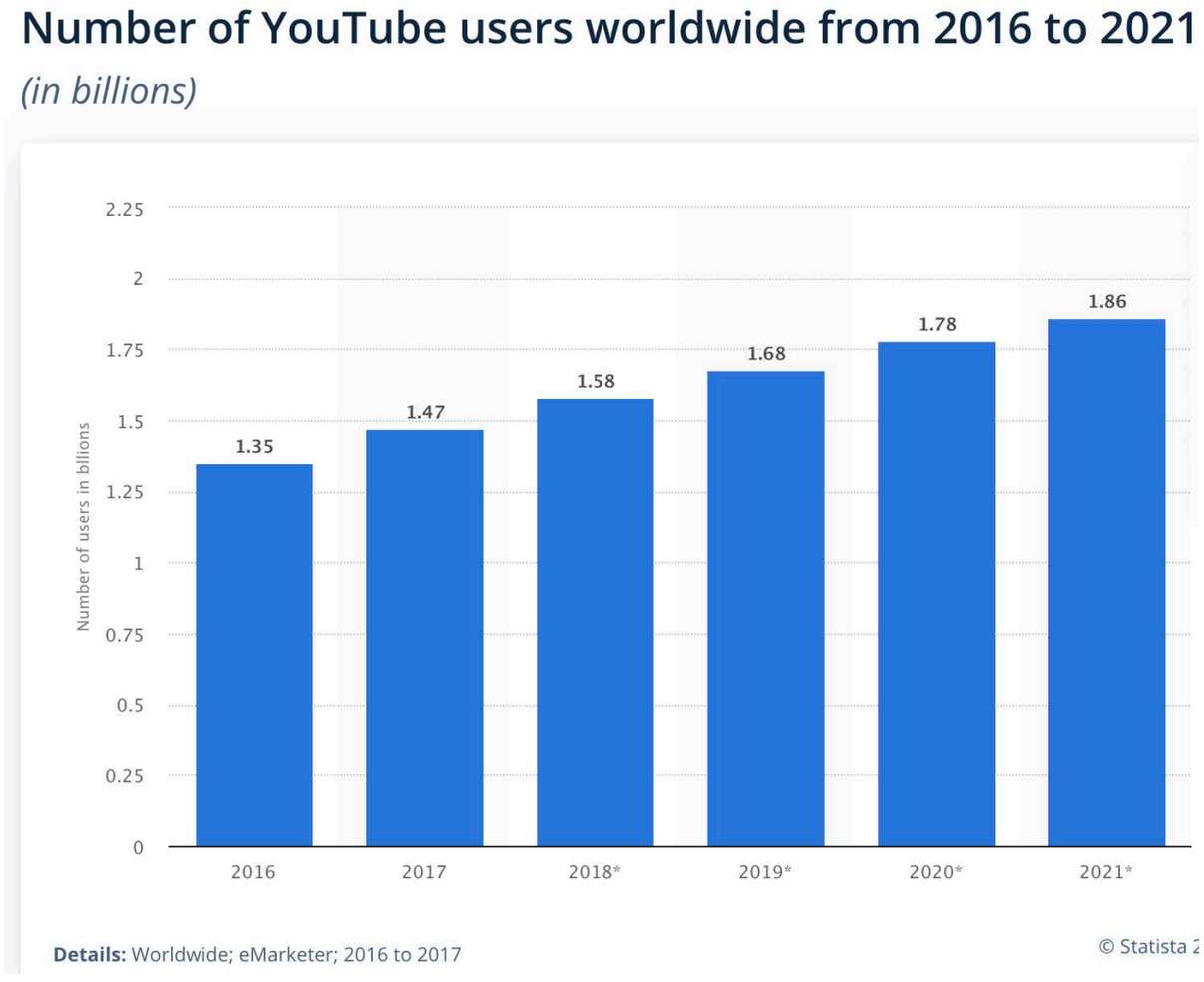
always more users

x

always more content

Most of the Internet traffic is video already and we keep watching more of more of them





Ever-increasing network traffic stresses network infrastructures

One recent example

somehow arguable though

Netflix Lowers Video Quality to Aid Broadband ISP Congestion

Thursday, March 19th, 2020 (8:53 pm) - Score 5,555

Email | Link News 43 Comments

In a somewhat unusual development Netflix, the global internet video streaming giant, has responded to some concerns about broadband congestion in other European countries by agreeing with the EU to lower the quality (bitrate) of its videos for the next 30 days (i.e. easing the strain on fixed line ISPs and mobile operators).

The move will result in the company reducing video bitrates across all of their streams, although they have yet to say if those paying extra for the highest quality are going to be compensated. Netflix have also yet to confirm if this will apply to the United Kingdom, where the majority of fixed line providers have yet to really struggle with managing increased demand (we've covered this here).

https://www.ispreview.co.uk/index.php/2020/03/netflix-lowers-video-quality-to-aid-broadband-isp-congestion.html

In this lecture, you'll learn

how to optimize the

- performance
- flexibility
- reliability

of network infrastructures

Techniques

Traffic Engineering

Load Balancing

Quality of Service

Fast Convergence

Techniques

Traffic Engineering

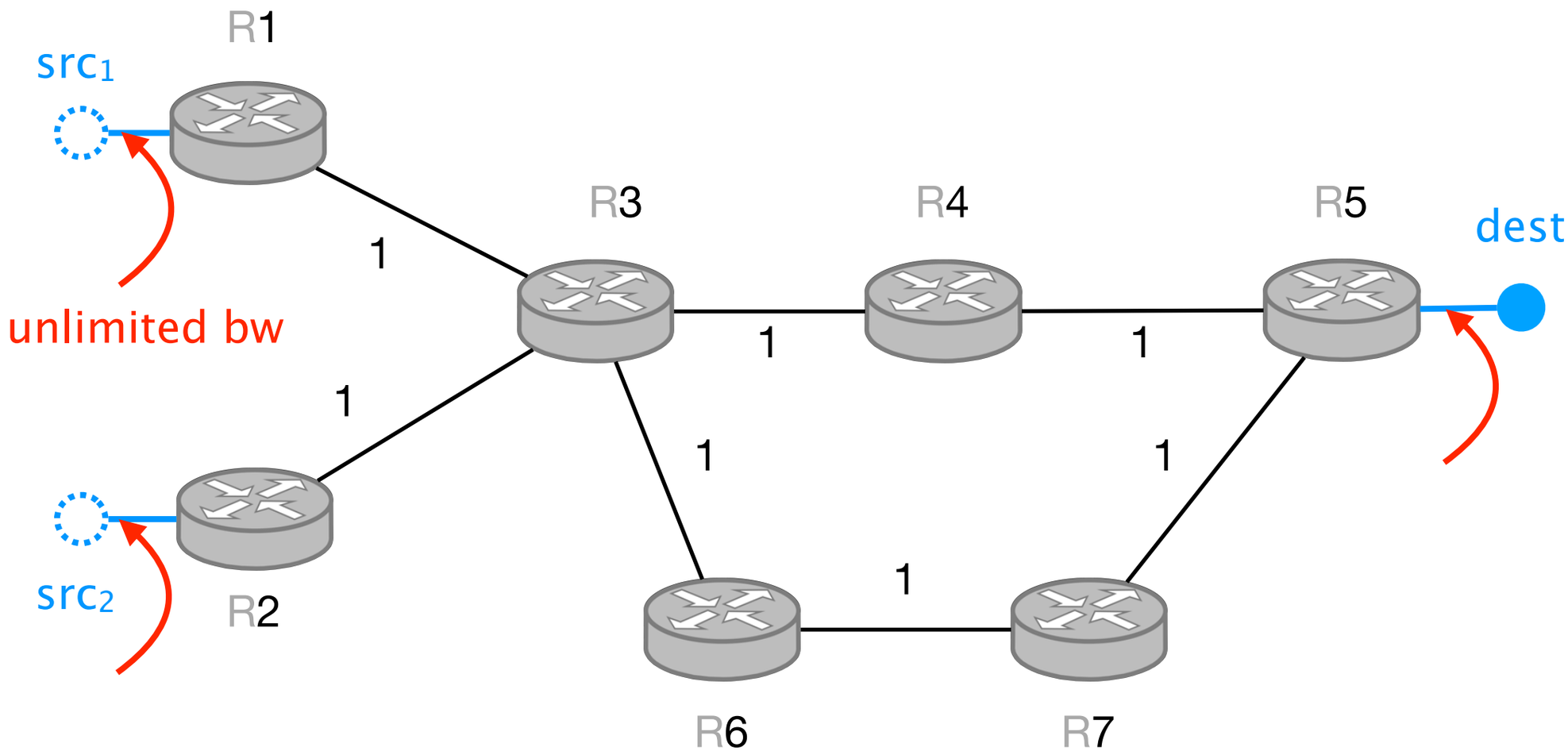
Load Balancing

Quality of Service

Fast Convergence

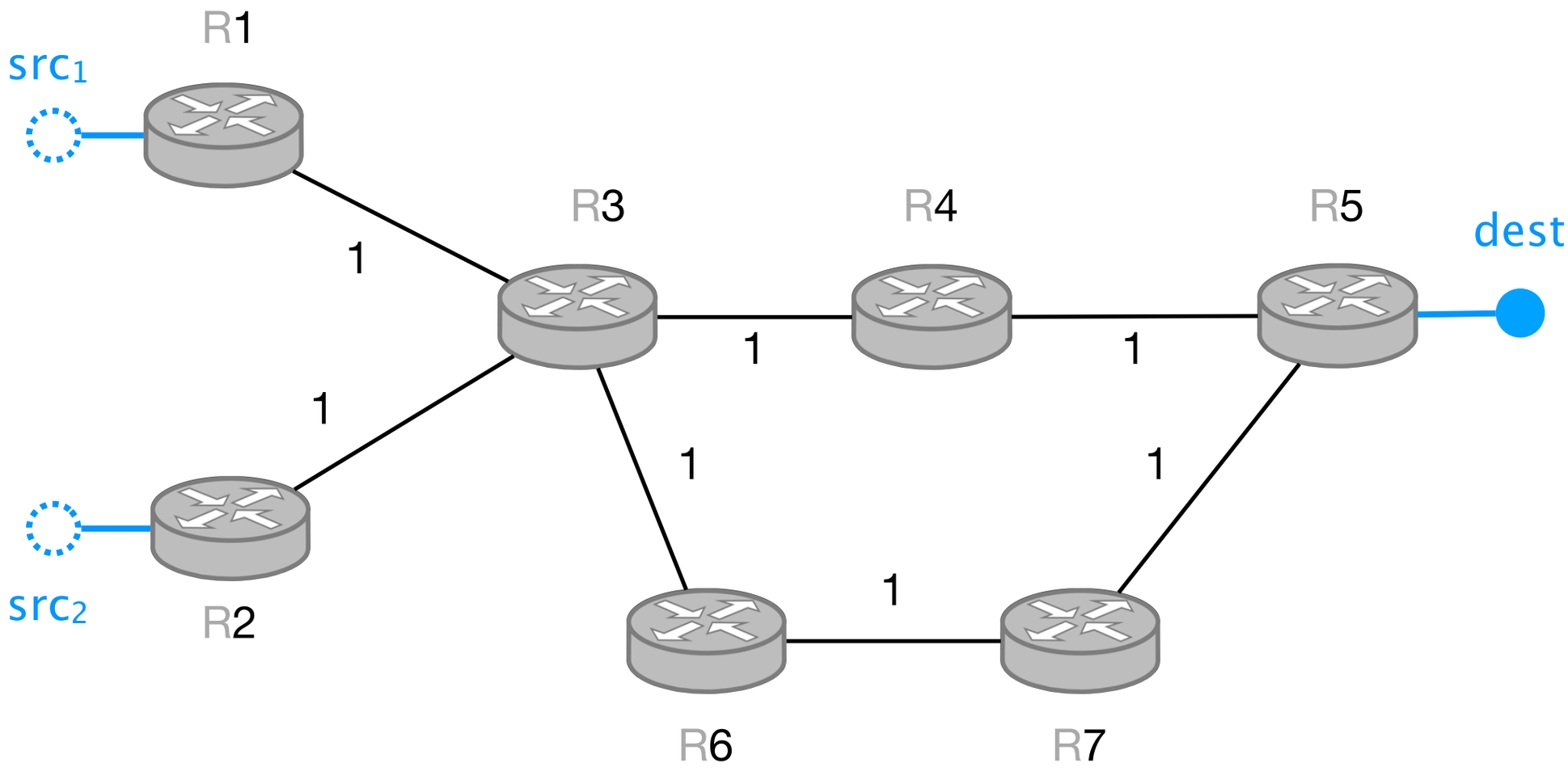
Consider this 10-Gbps network running vanilla OSPF

src_1 and src_2 send traffic to dest, 1 TCP flow each



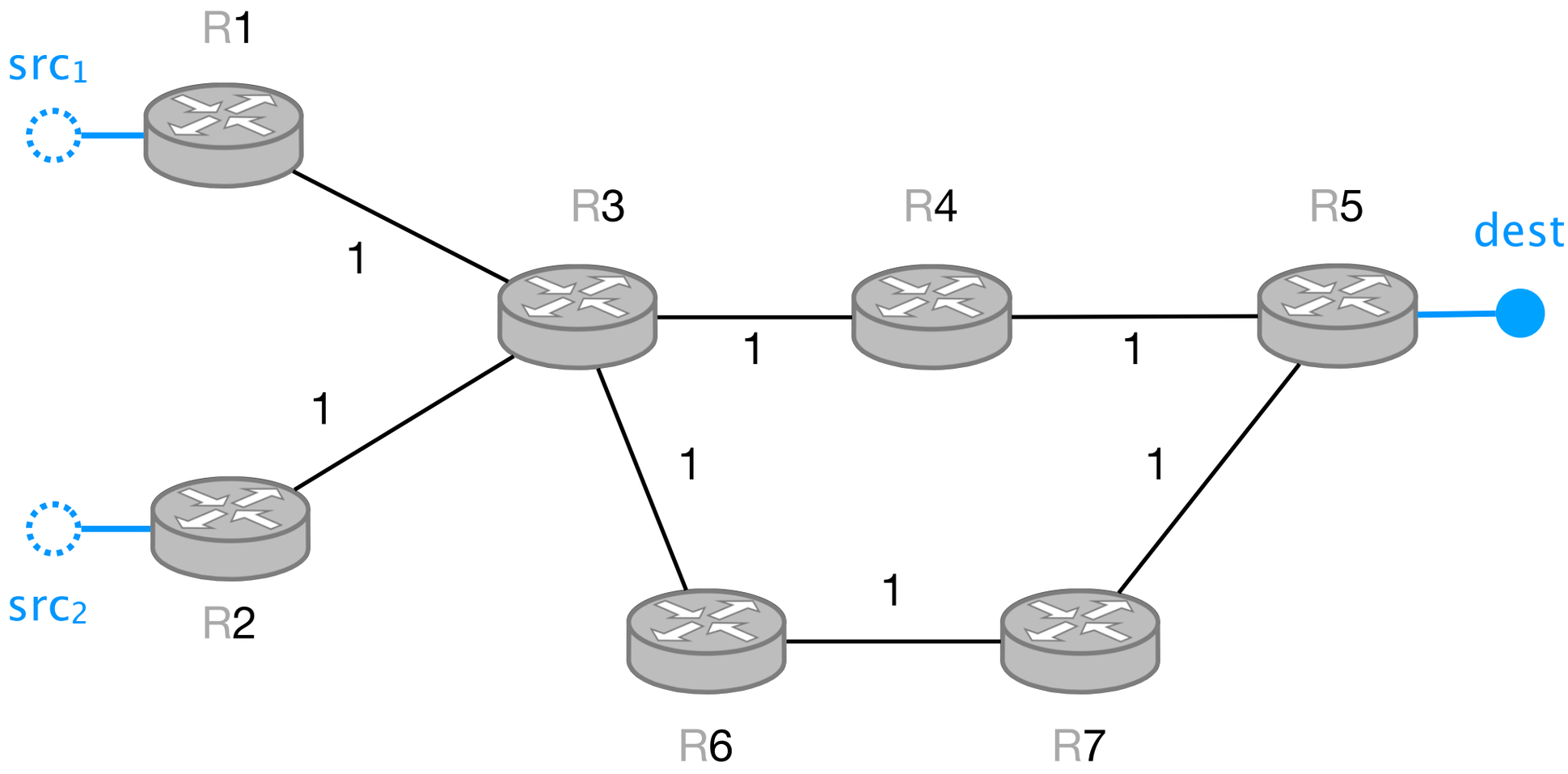
Consider this 10-Gbps network running vanilla OSPF

src_1 and src_2 send traffic to dest, 1 TCP flow each



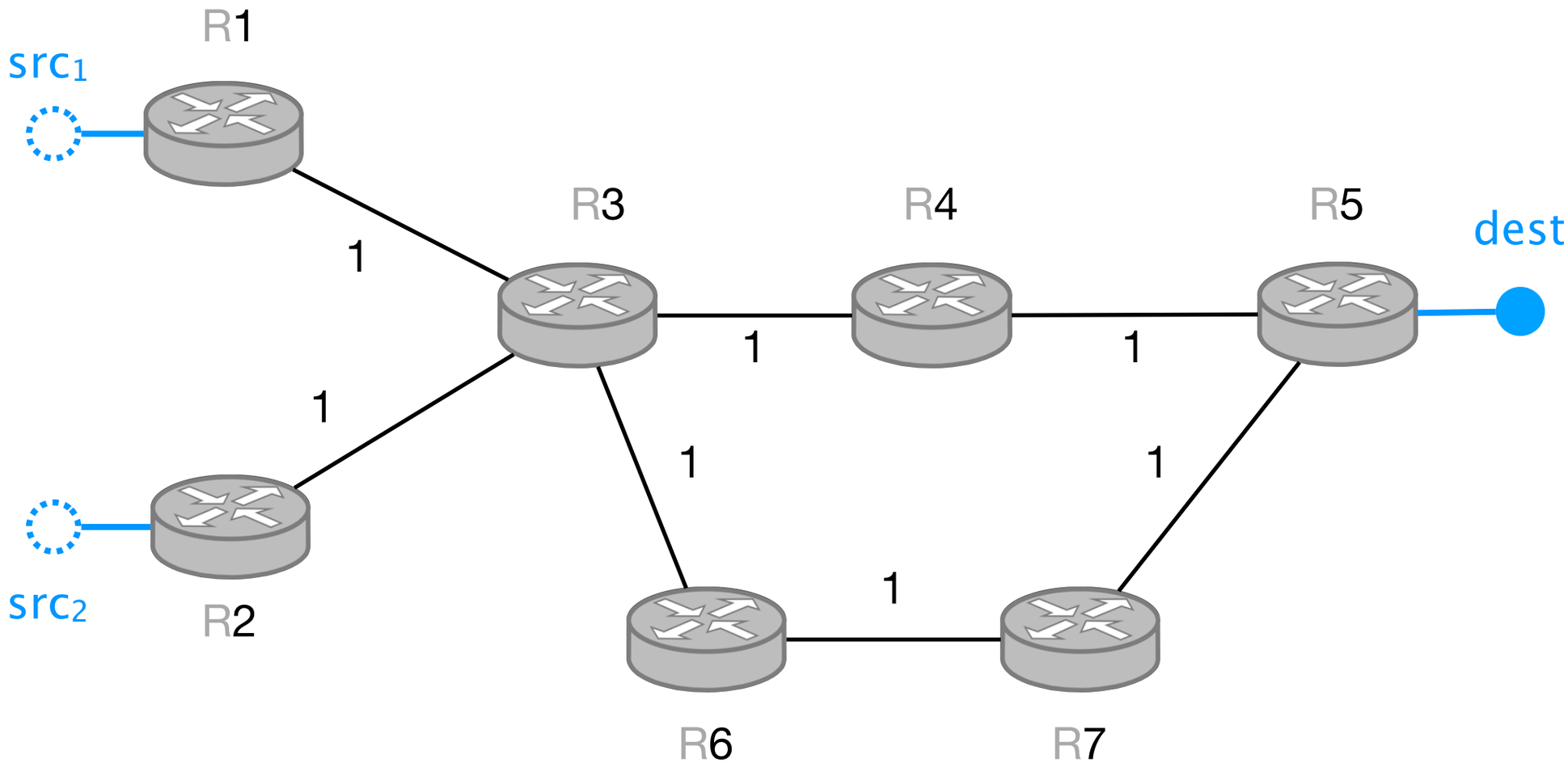
What's the max. throughput (src_1, dest) and (src_2, dest) can achieve?

What can we do better?



What can we do better?

What about better weights?



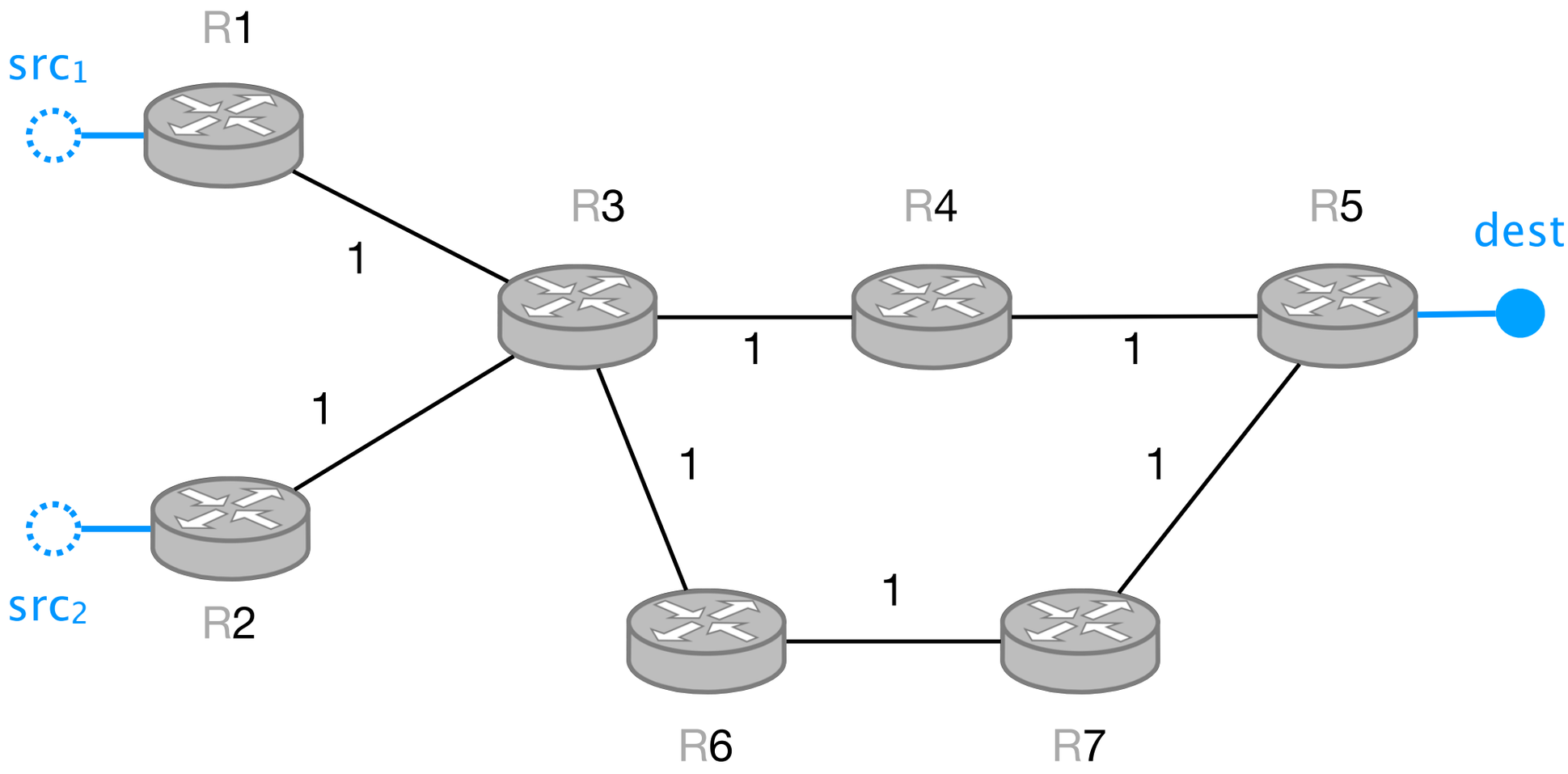
Equal-Cost Multi-Path (ECMP) is a routing strategy in which IP routers splits traffic over all best paths

Equal-Cost Multi-Path (ECMP) is a routing strategy in which IP routers splits traffic over all best paths

in OSPF, best == shortest

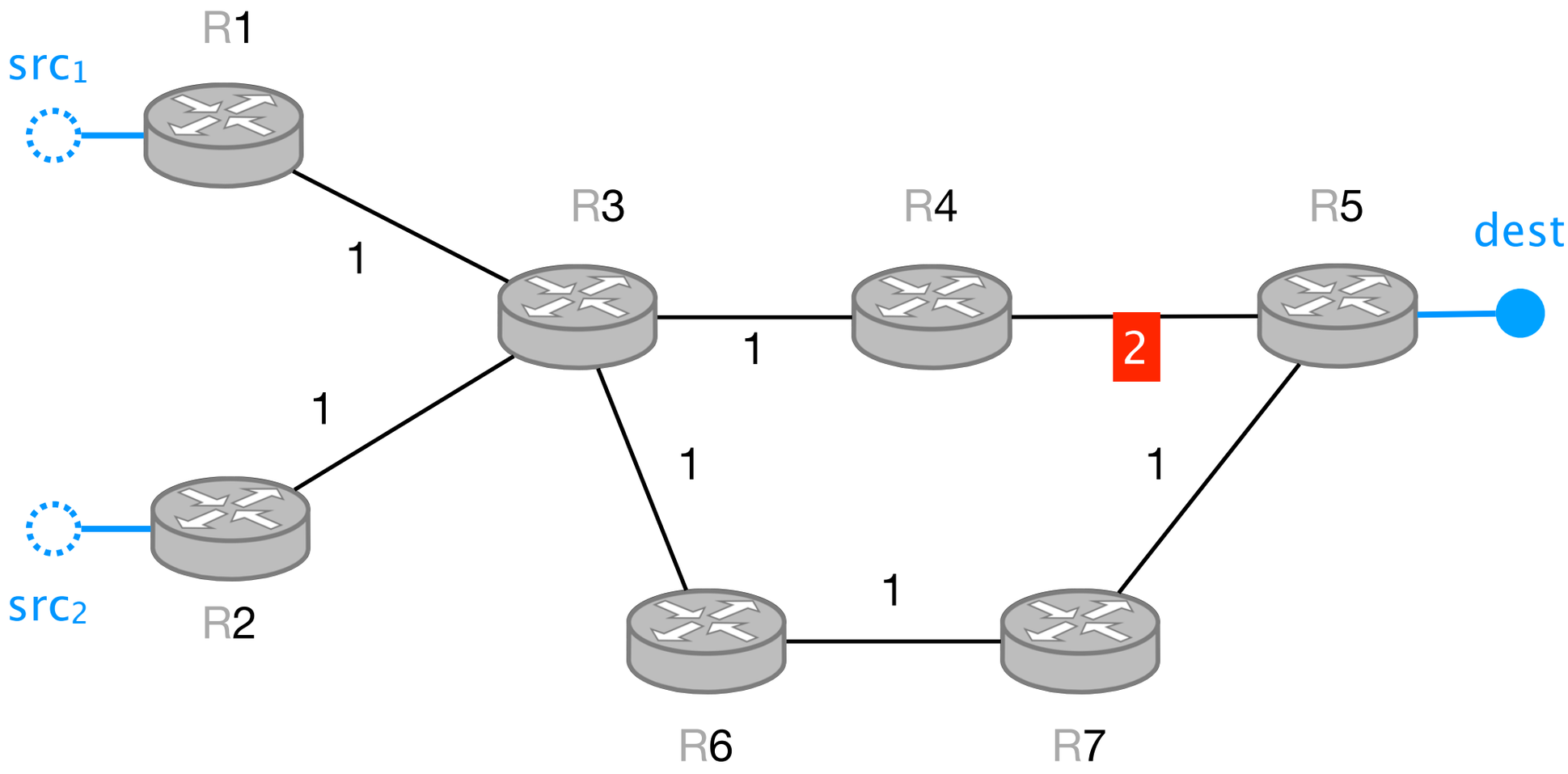
What can we do better?

What about better weights?



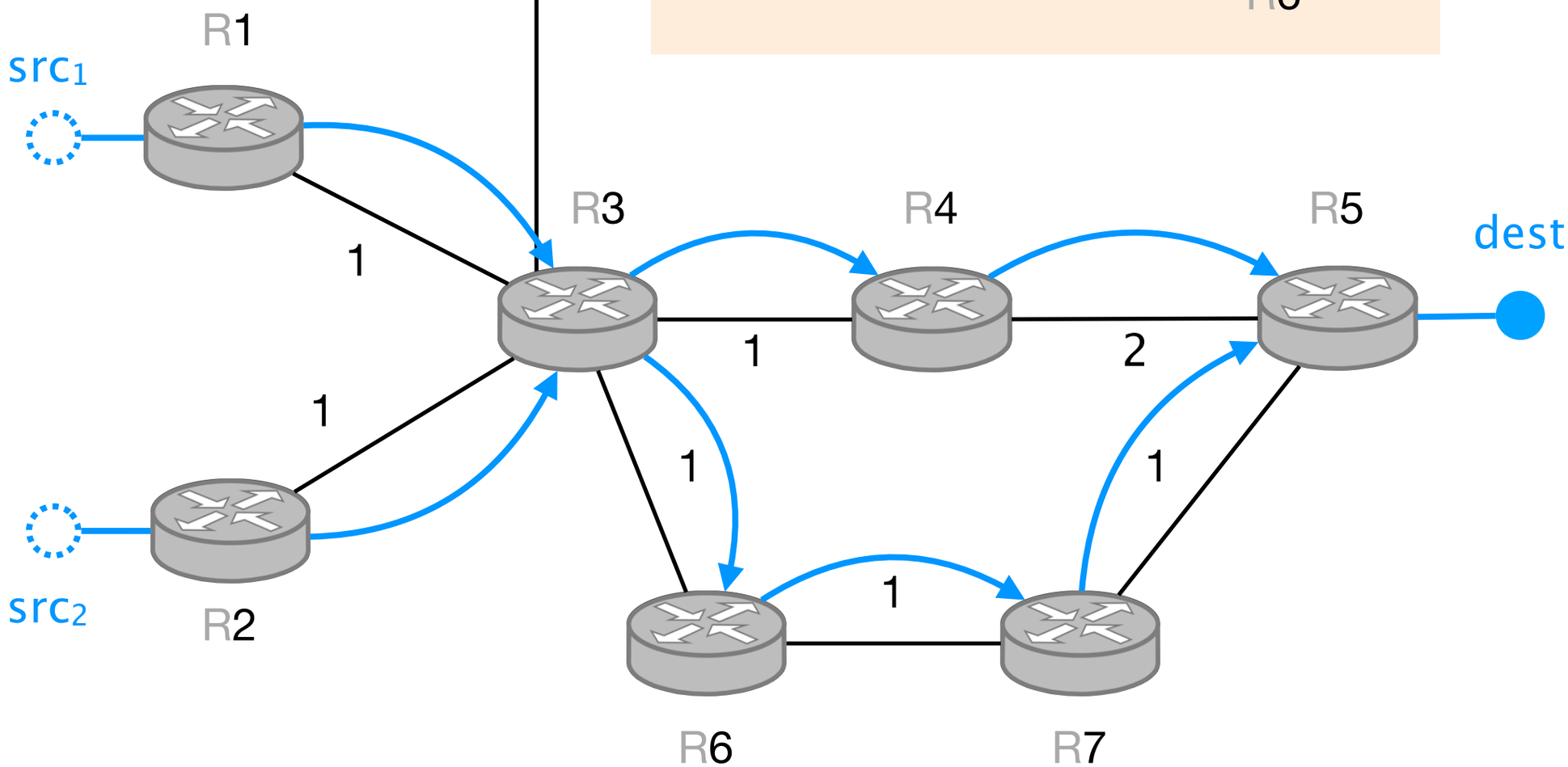
What can we do better?

What about better weights?



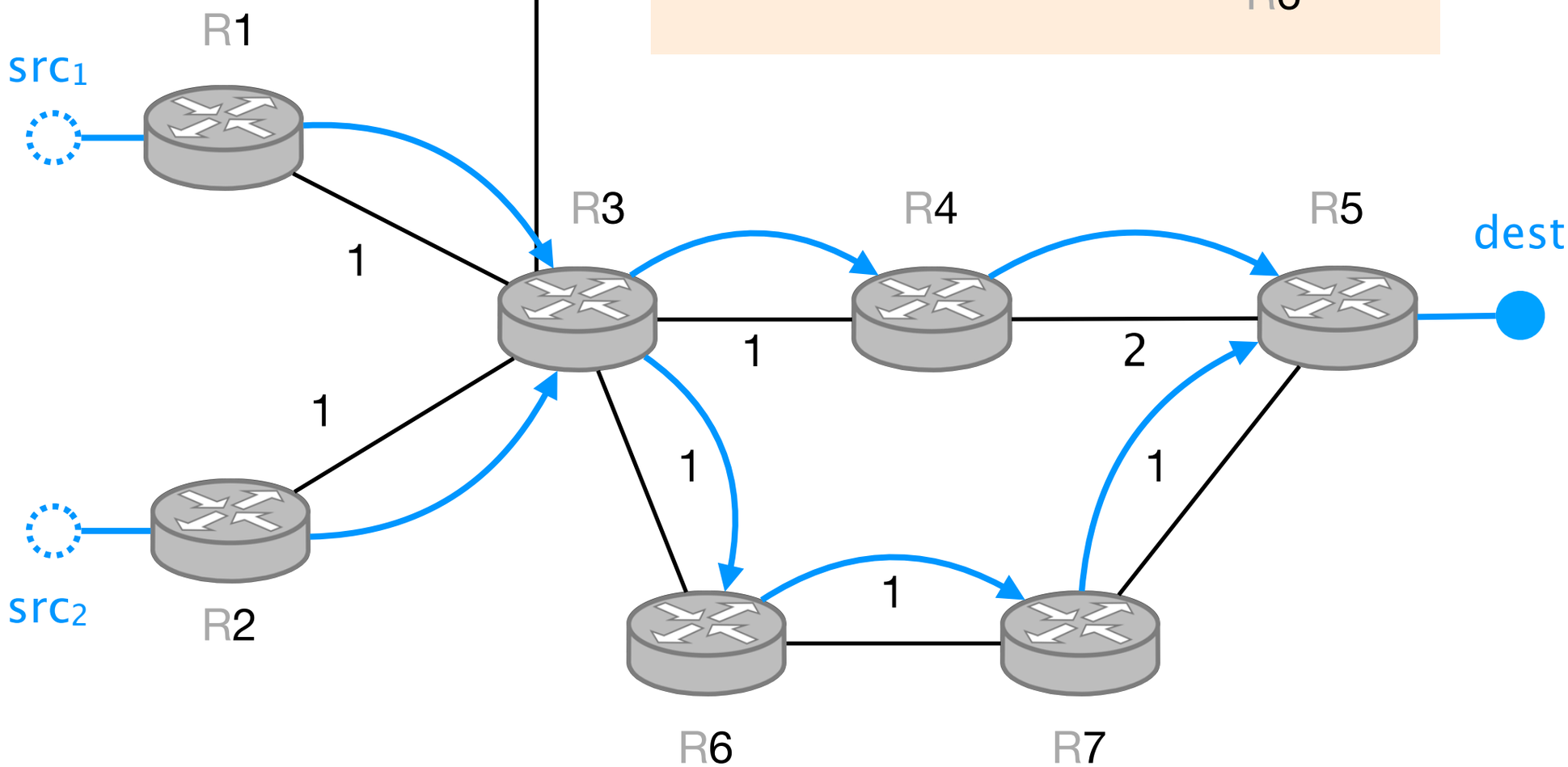
R3's routing table

pfx	min_cost	next_hop
dest	3	R4
		R6



R3's routing table

pfx	min_cost	next_hop
dest	3	R4
		R6

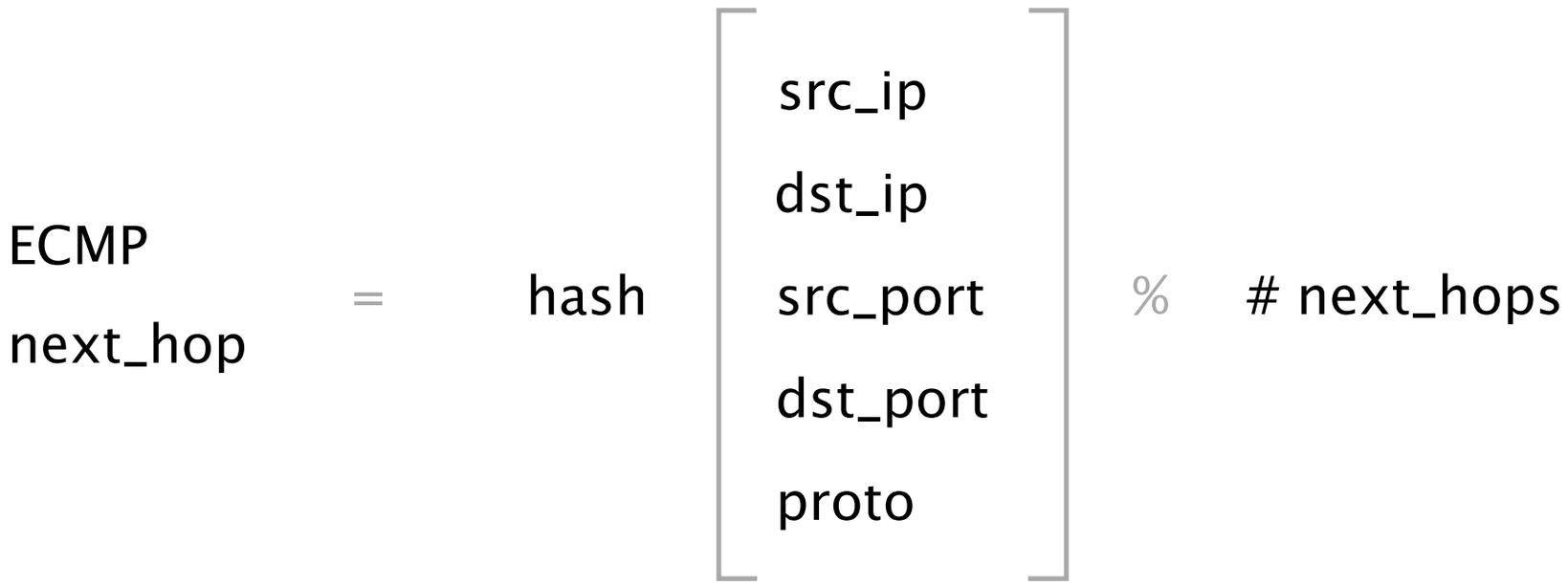


Is it enough to guarantee we fully utilize the network? Nope…

Equal-Cost Multi-Path (ECMP) is a routing strategy in which IP routers splits traffic over all best paths

how?

Equal-Cost Multi-Path (ECMP) relies on *stateless* hash functions to split traffic



Property

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

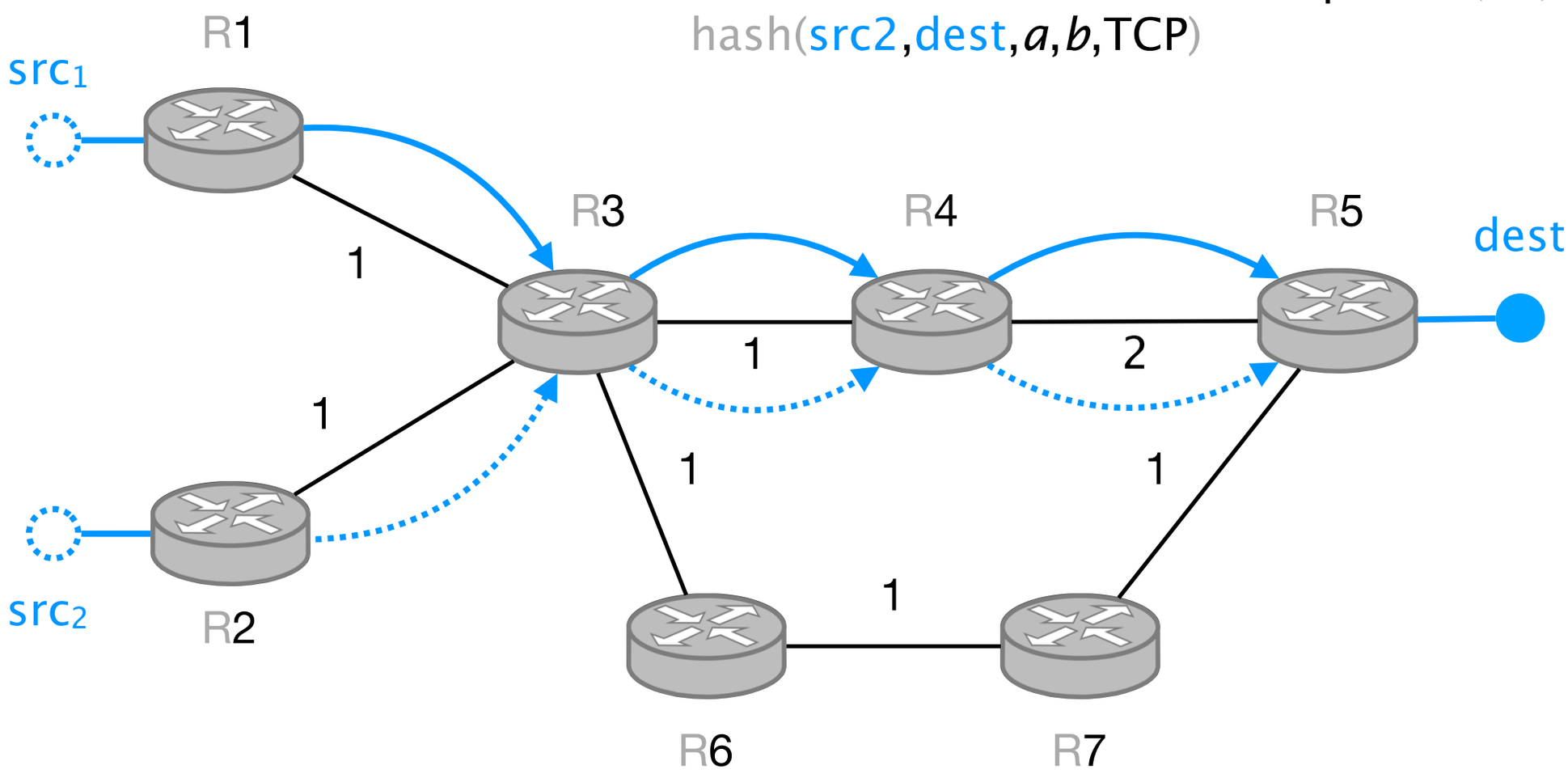
(doing so avoids reordering)

What can go wrong?

Hash collisions!

hash(src1,dest,*x*,*y*,TCP) = port 0 (R4)

hash(src2,dest,*a*,*b*,TCP)



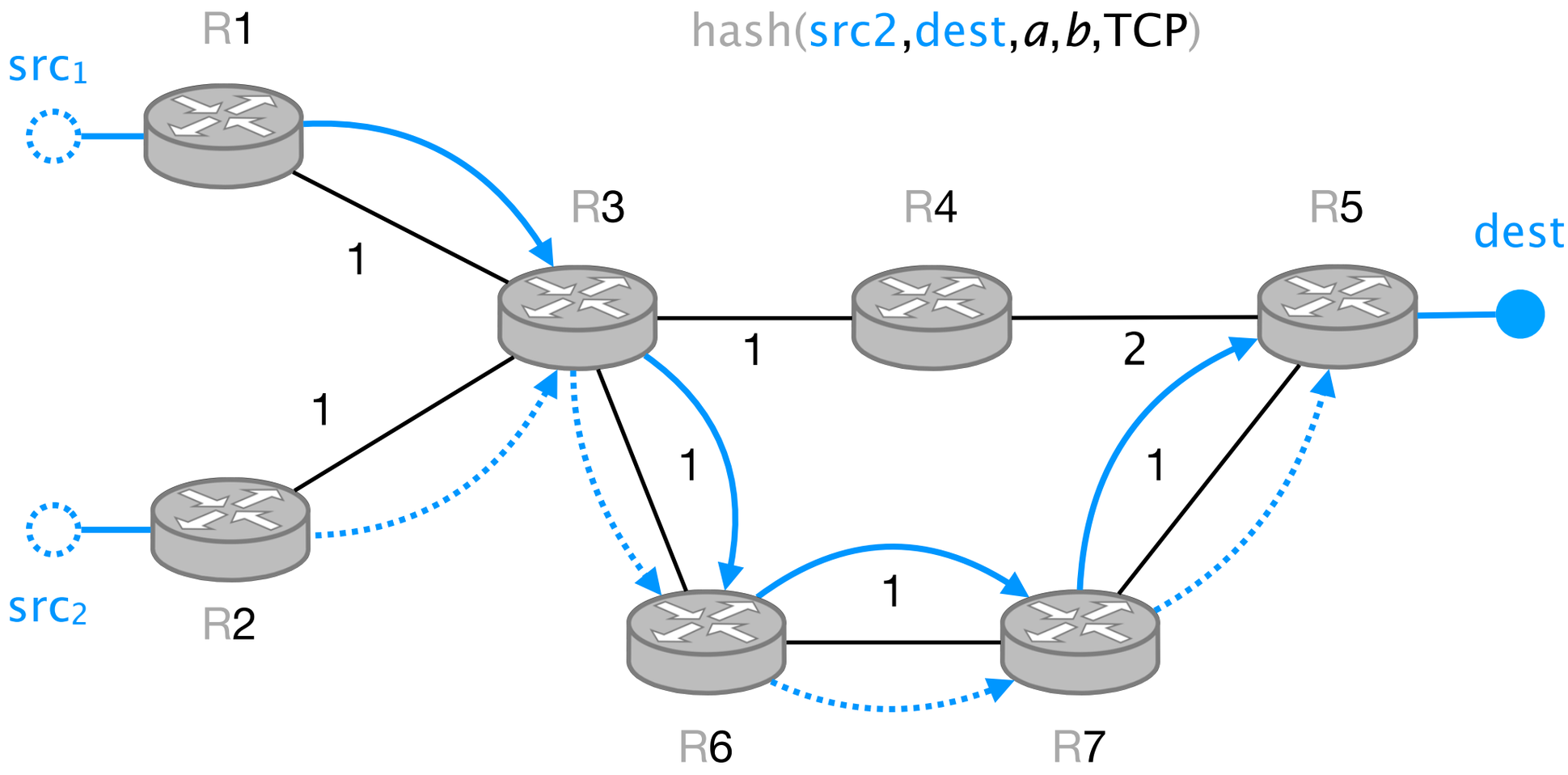
What can go wrong?

Hash collisions!

hash(src1,dest,*x*,*y*,TCP)

hash(src2,dest,*a*,*b*,TCP)

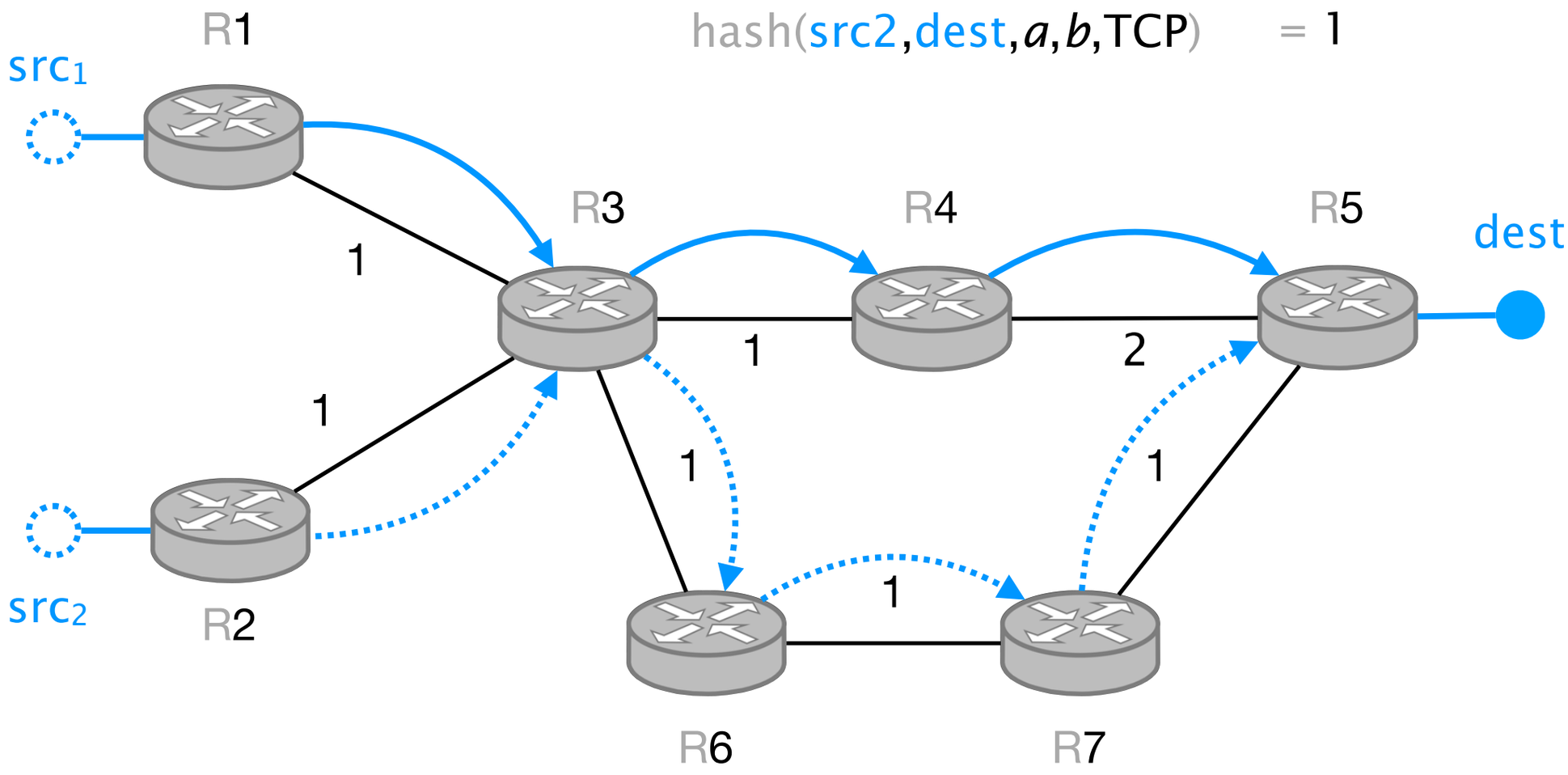
= port 1 (R6)



Traffic is effectively load balanced only when the 2 hashes don't match

hash(src1,dest,*x*,*y*,TCP) = 0 (or vice-versa)

hash(src2,dest,*a*,*b*,TCP) = 1



We'll see different ways to solve this problem including:

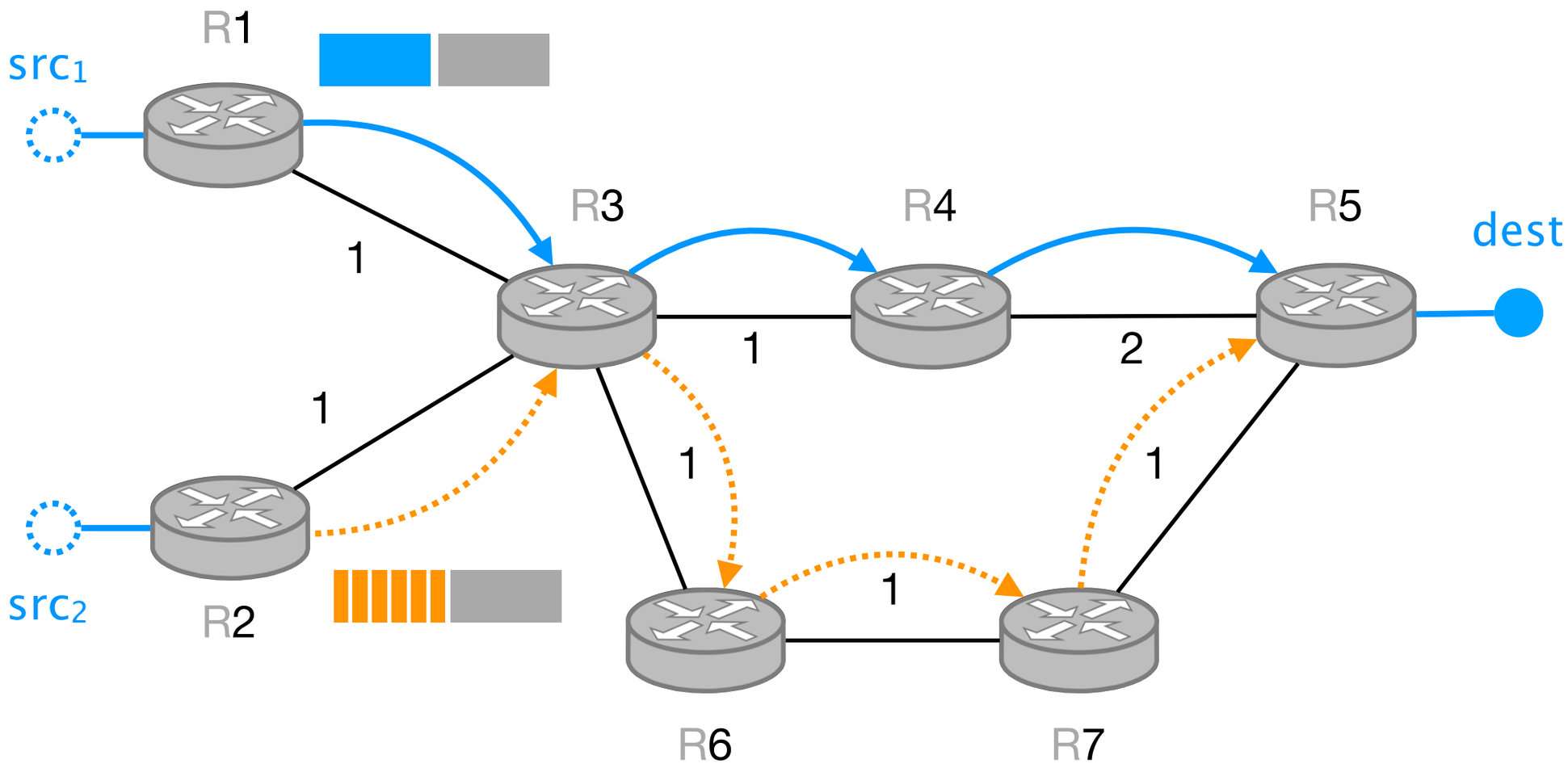
smarter load-balancing functions and label switching

We'll see different ways to solve this problem including:
smarter load-balancing functions and label switching

next_hop = function

- queue occupancy
- downstream path load
- packet interarrival time
- random
- ...

We'll see different ways to solve this problem including:
smarter load-balancing functions and label switching



Techniques

Traffic Engineering

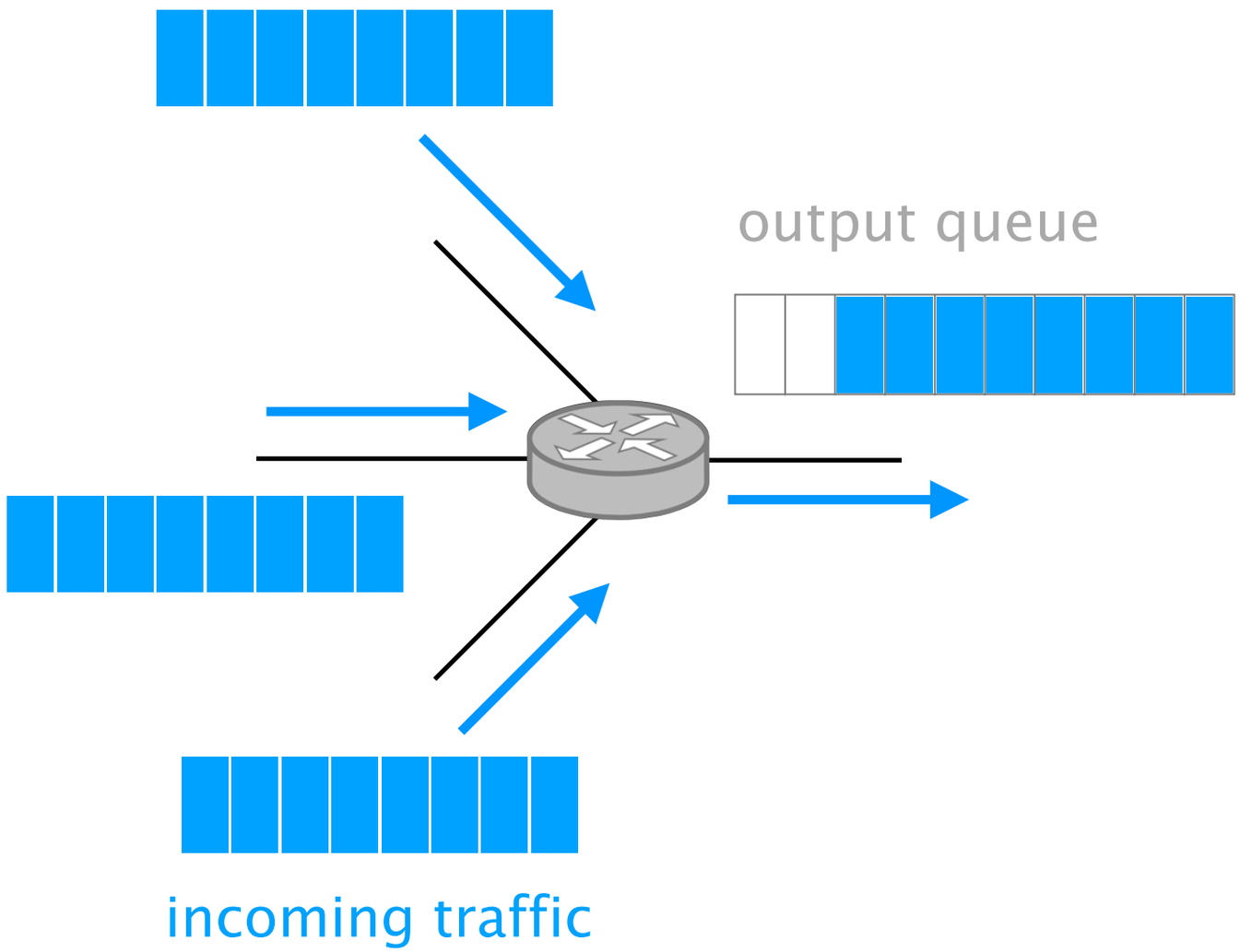
Load Balancing

Quality of Service

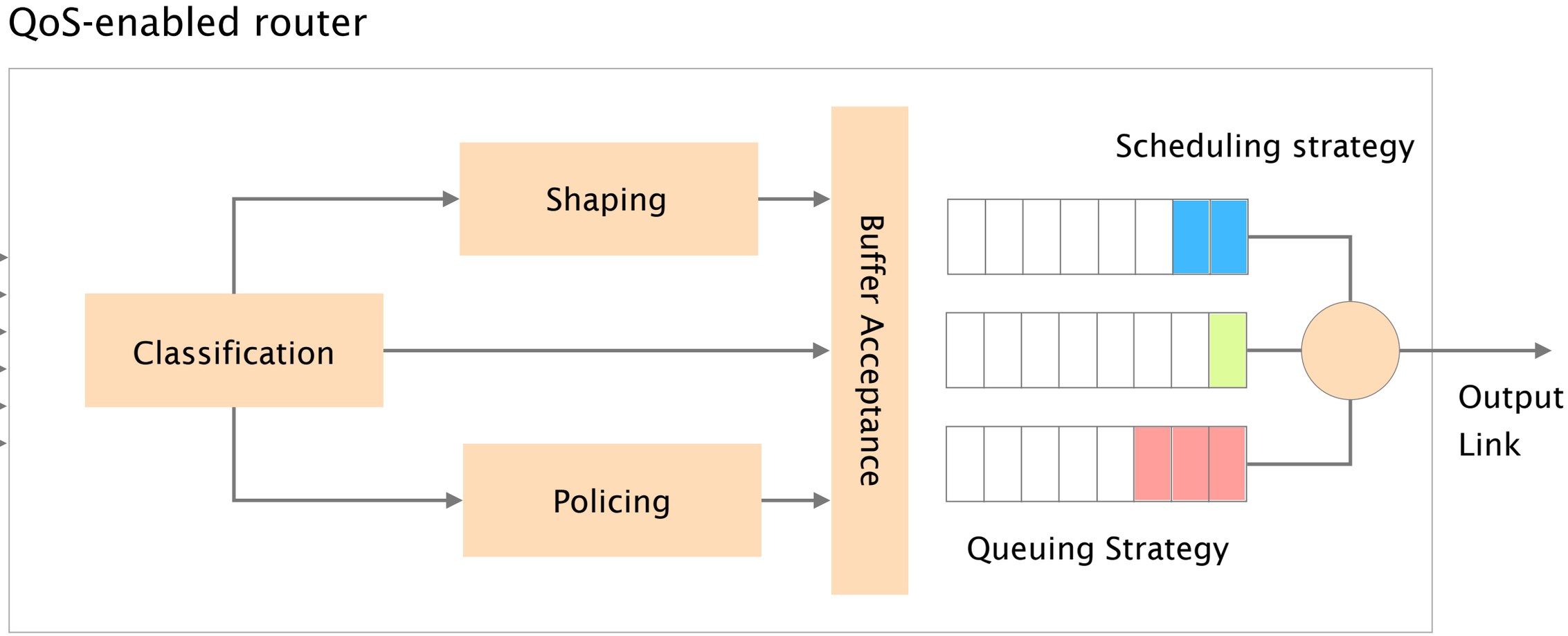
Fast Convergence

While traffic engineering and load balancing help,

they cannot *always* prevent congestion



We'll see different ways to manage congestion using Quality of Service (QoS)



Techniques

Traffic Engineering

Load Balancing

Quality of Service

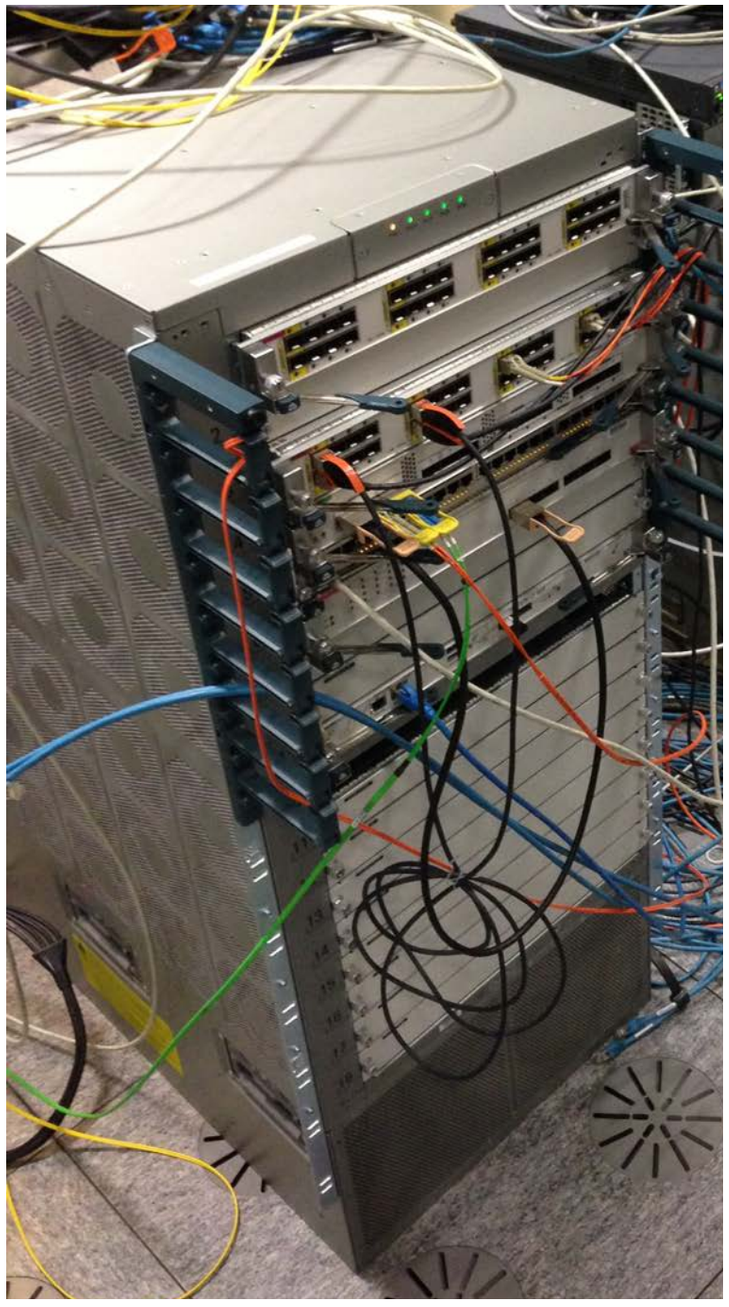
Fast Convergence

IP networks converge slowly upon failures

at least, by default

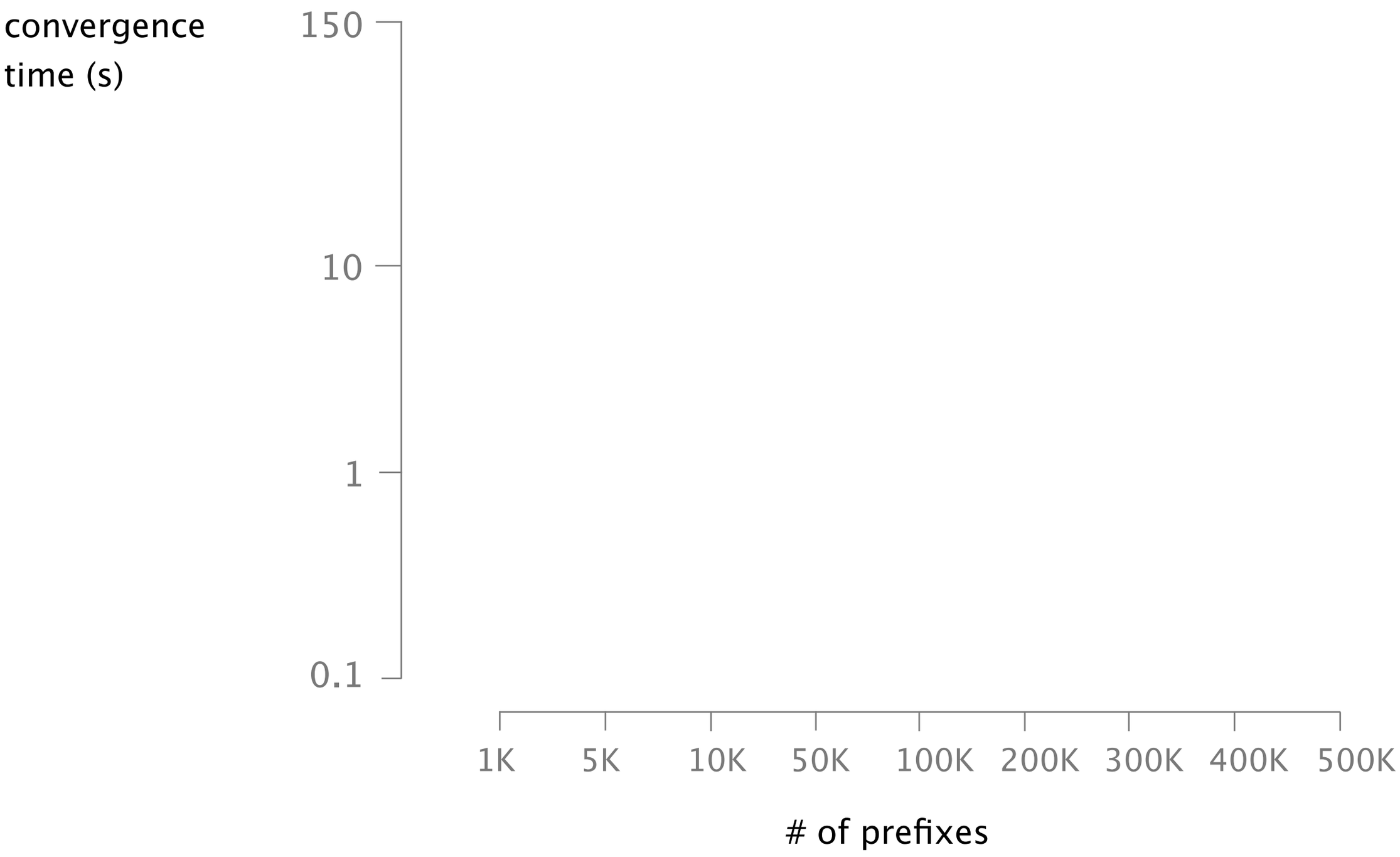
We measured how long it takes for

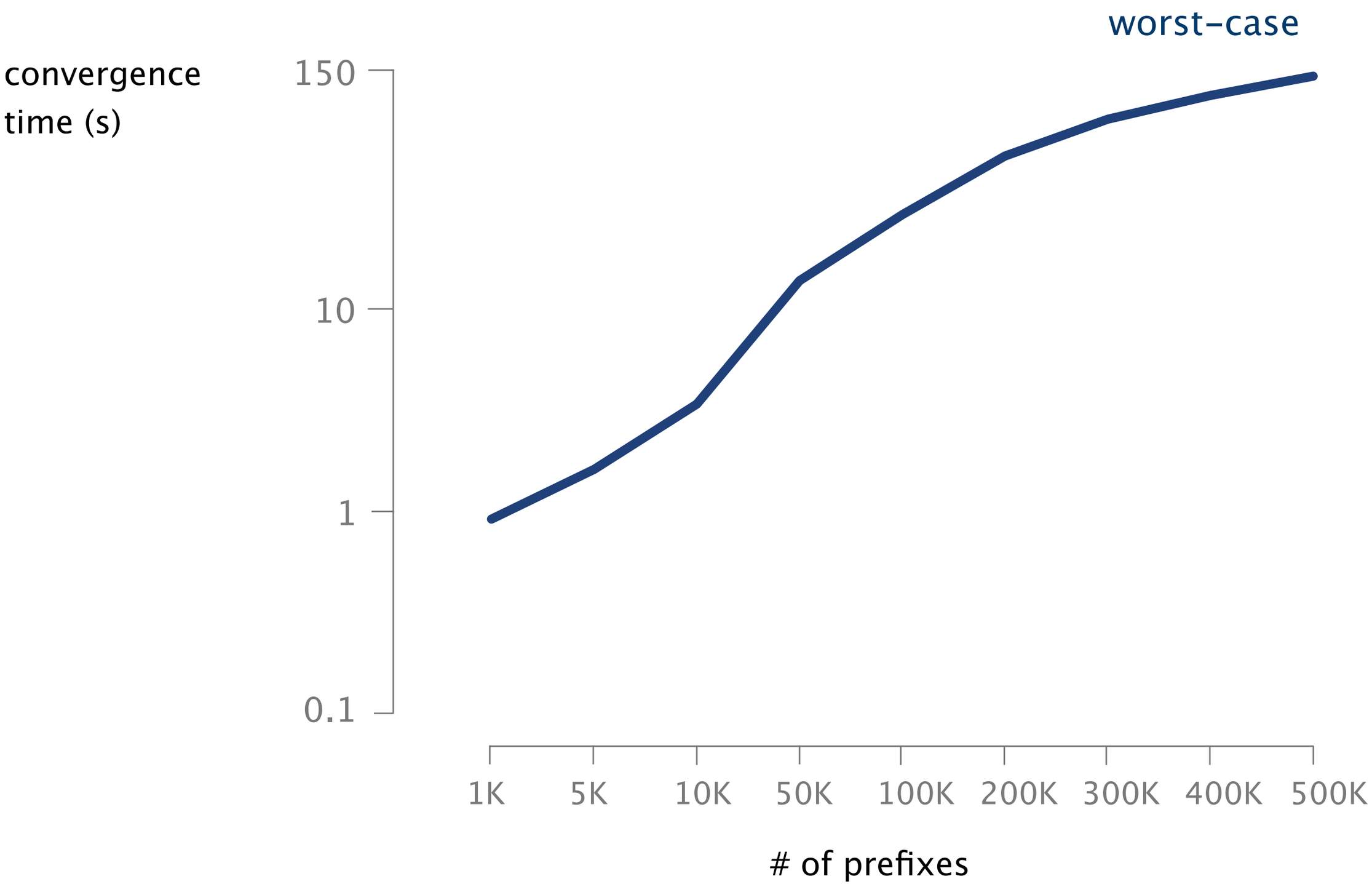
a former ETH router to converge

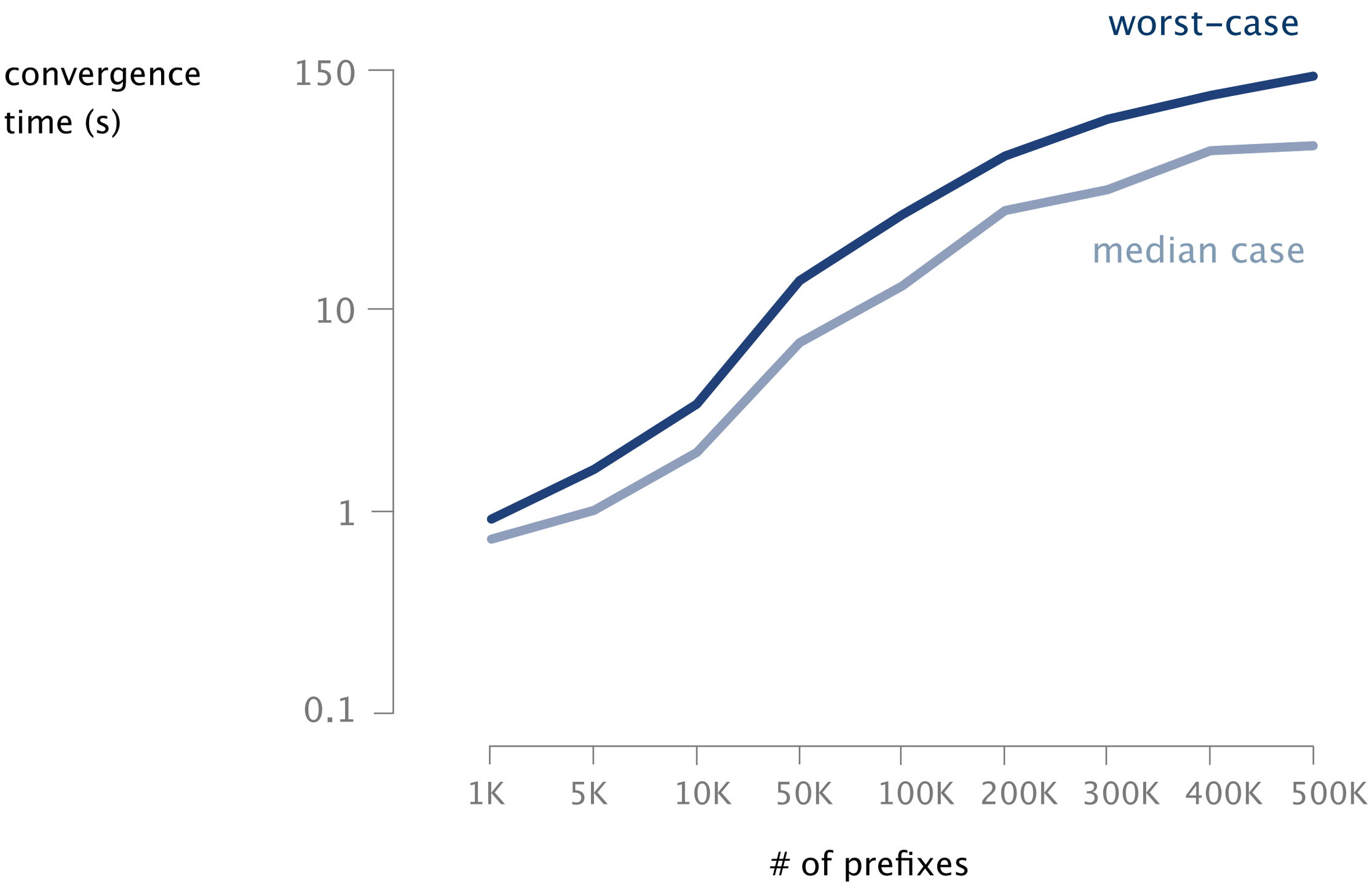


Cisco Nexus 9k

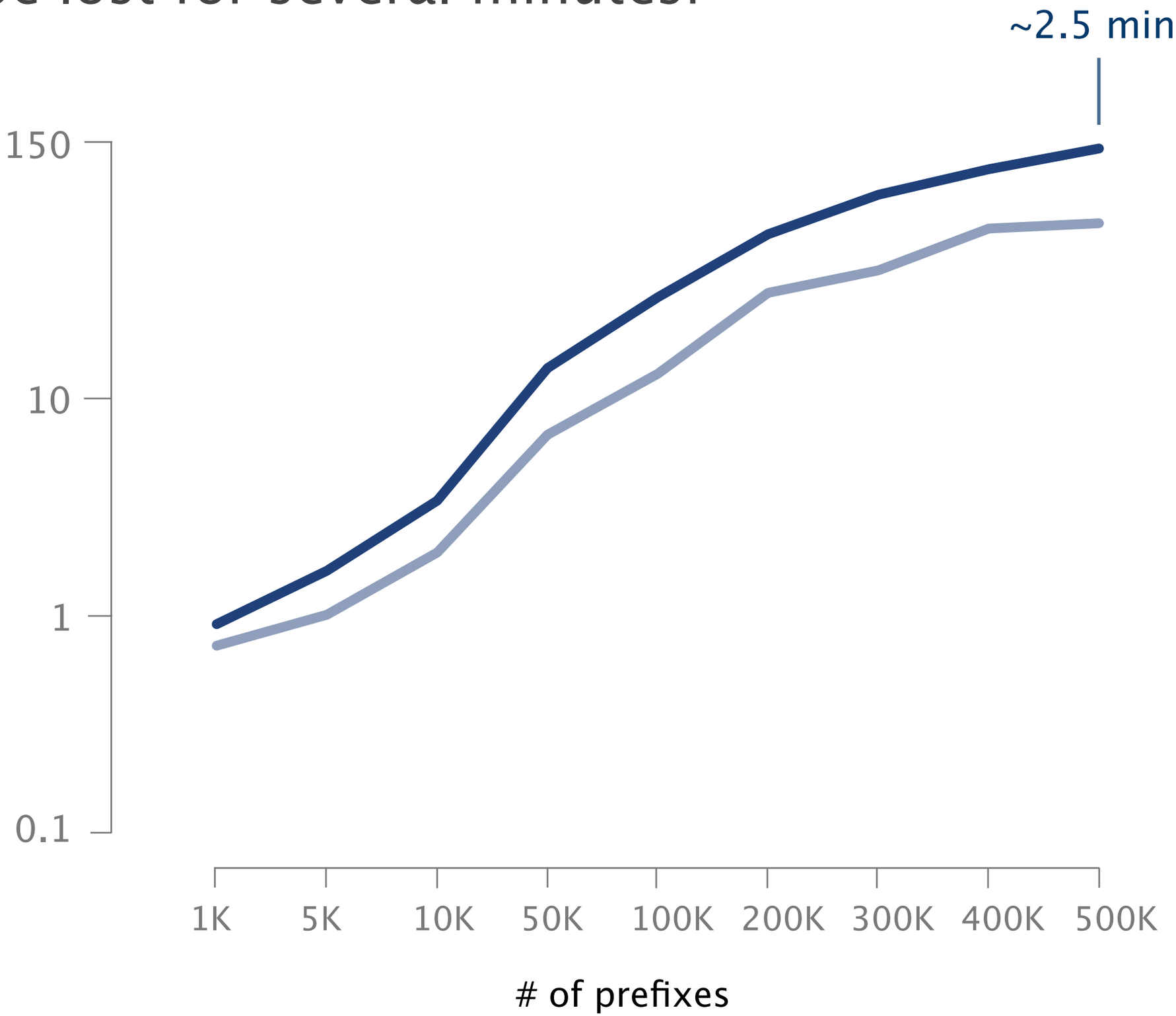
ETH's routers


convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes


worst-case
convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes


worst-case
median case
convergence
time (s)
150
10
1
0.1
1K
5K
10K
50K
100K
200K
300K
400K
500K
# of prefixes

Traffic can be lost for several minutes!



What takes *so* long?

detect the failure	<10 ms
report the failure	<10 ms
generate/flood routing messages	10s of ms
recompute routing paths	10s of ms
communicate next-hops to the line cards	100s of ms
install new next-hops	10s of ms

What takes *so* long?

	detect the failure	<10 ms
	report the failure	<10 ms
	generate/flood routing messages	10s of ms
	recompute routing paths	10s of ms
bottleneck	communicate next-hops to the line cards	100s of ms
	install new next-hops	10s of ms

We'll see different Fast Reroute technologies
to speed up convergence time

Fast Reroute
technologies

detect the failure

report the failure

generate/flood routing messages

recompute routing paths

communicate next-hops to the line cards

install new next-hops

hw acceleration

pre-computation

pre-provisionning
fast activation

max. convergence time for *any* failure

<1 sec

Techniques

Traffic Engineering

Load Balancing

Quality of Service

Fast Convergence

Besides learning about these techniques,

you'll learn how to *implement* them

Besides learning about these techniques,

you'll learn how to *implement* them

implement

your own forwarding logic

in programmable networks

Until the early 2010s,

network devices tended to be completely locked down

closed software

closed hardware

Cisco™ device

Since then

networks have become *programmable*

Controller

open-source software

ONOS®
Open Network Operating System

standardized interface

OpenFlow™

reprogrammable hardware

P4

Network programmability has attracted tremendous industry interest…

VMware Acquires Once-Secretive Start-Up Nicira for $1.26 Billion

JULY 23, 2012 AT 1:25 PM PT

VMware, the software company best known for its virtualization technology that forms the backbones of so-called cloud computing today, said it will pay $1.26 billion for Nicira, a networking start-up that has sought to do to networks what VMware has done to computers.

The news comes on the same day that VMware was to report quarterly earnings. And while I don't usually cover VMware's earnings, I may as well mention the results: The company reported revenue for the quarter ended June rose to $1.12 billion, while earnings on a per-share basis were 68 cents. Analysts had been expecting sales of $1.12 billion and earnings of 66 cents.

Nicira had been running in stealth mode for quite awhile; I got to reveal its plans to the world last February.

The deal amounts to a nice payoff for Nicira's investors including Andreessen Horowitz, Lightspeed Venture Partners and NEA, as well as VMware founder Diane Greene and venture capitalist Andy Rachleff.

With $600M Invested in SDN Startups, the Ecosystem Builds

Scott Raynovich, June 10, 2014

More than $600 million has been invested in at least two dozen software-defined networking (SDN) startups so far, according to Rayno Report research. You can expect that to continue to climb. With the SDN ecosystem starting to take hold with a broad range of alliances and distribution partnerships, we're just getting started.

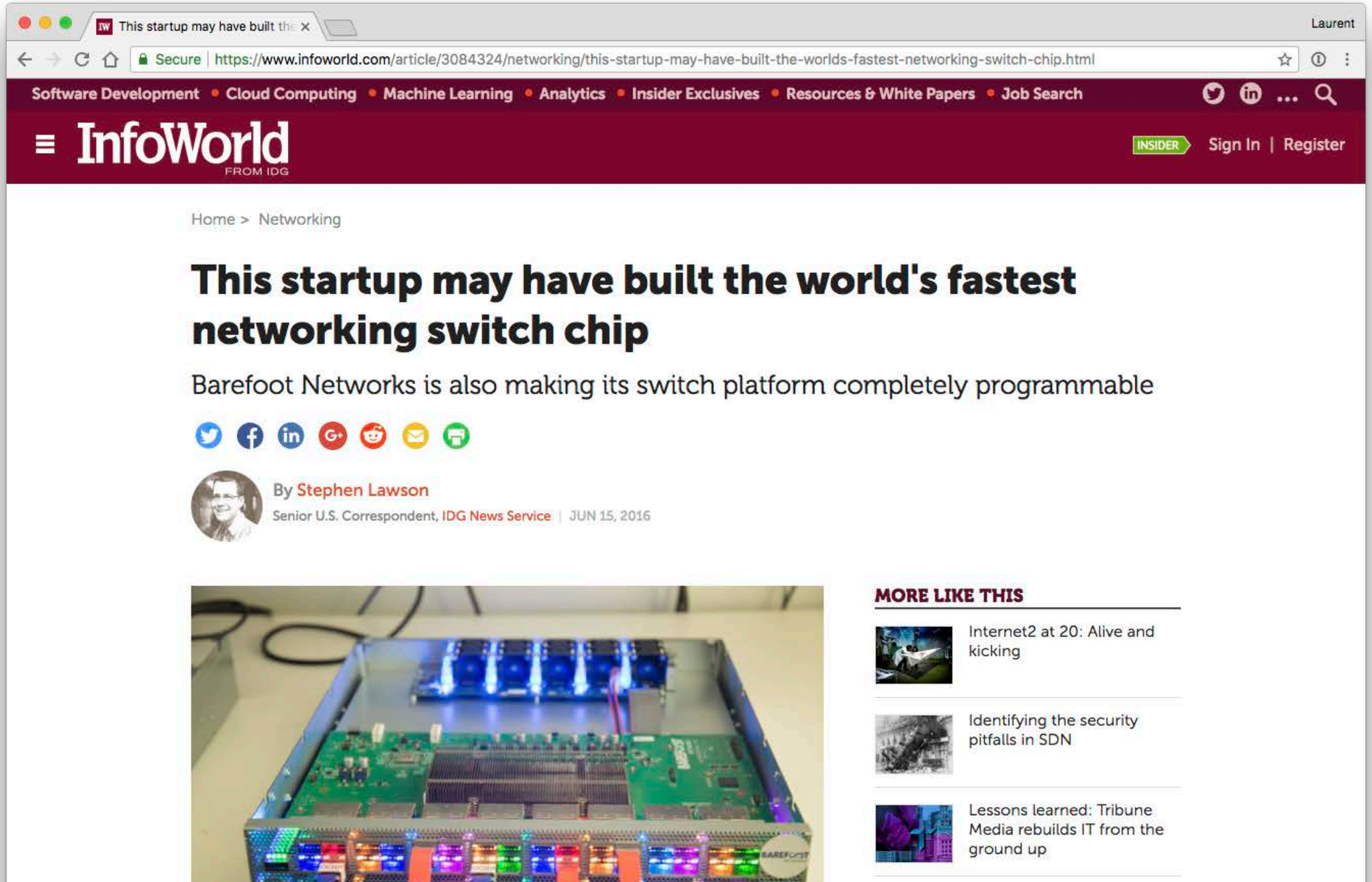
The Arista IPO will help build visibility for next-generation, software-driven networking. But Arista is selling its own hardware and is not an SDN pure-play. A new line of SDN startups, with a more radical approach to software-based networking, is building momentum. These newer SDN startups are just getting their gear into customers' hands and starting to build sales channels, so you can expect a long revenue ramp.

This excitement is boosting startup valuations, according to Rayno Report research. There are now at least ten SDN startups with valuations over $100 million. As I reported in April, a recent investment in Cumulus Networks pushed up the valuation of the private company north of $300 million, according to industry sources. Big Switch, which did a deal in 2012 valuing it near $170 million, took money from Intel in 2013, most likely boosting its valuation to over $200 million, according to several sources.

Related Articles

- How to Effectively Embed SDN in the Enterprise
- NFV and SDN: What's the Difference Two Years Later?
- sFlow Creator Peter Phaal On Taming The Wilds Of SDN & Virtual Networking
- Featured Article: Bringing Data-Driven SDN to the Network Edge
- NFV Delivers Pervasive Intelligence for MNOs

Barefoot Networks (Stanford startup) started to produce re-programmable network hardware in 2013



In June 2019,

Barefoot Networks was acquired by Intel

THE WALL STREET JOURNAL.

Europe Edition ▾ | September 22, 2019 | Print Edition | Video

Home World U.S. Politics Economy Business Tech Markets Opinion Life & Arts Real Estate WSJ. Magazine

SHARE

AA TEXT

TECH

Intel Agrees to Acquire Networking Startup Barefoot Networks

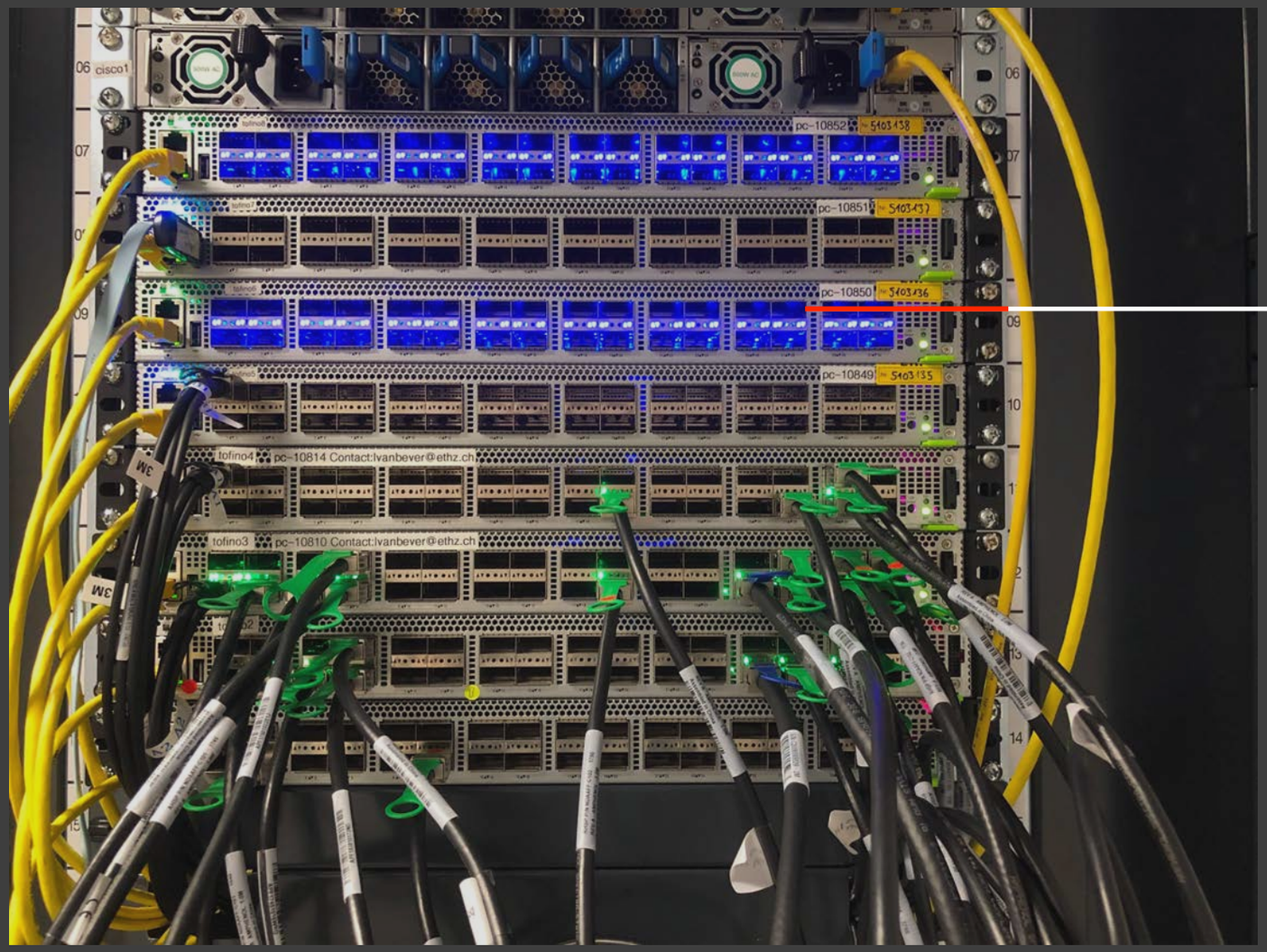
Barefoot Networks is backed by Google, Alibaba, Tencent and Goldman Sachs

P4 is a domain-specific language which specifies how a switch forwards packets

https://p4.org

A sneak peek at our own networking lab here @ETHZ

8x Tofino Wedge100BF-32X. Total capacity: 25.6 Tbps

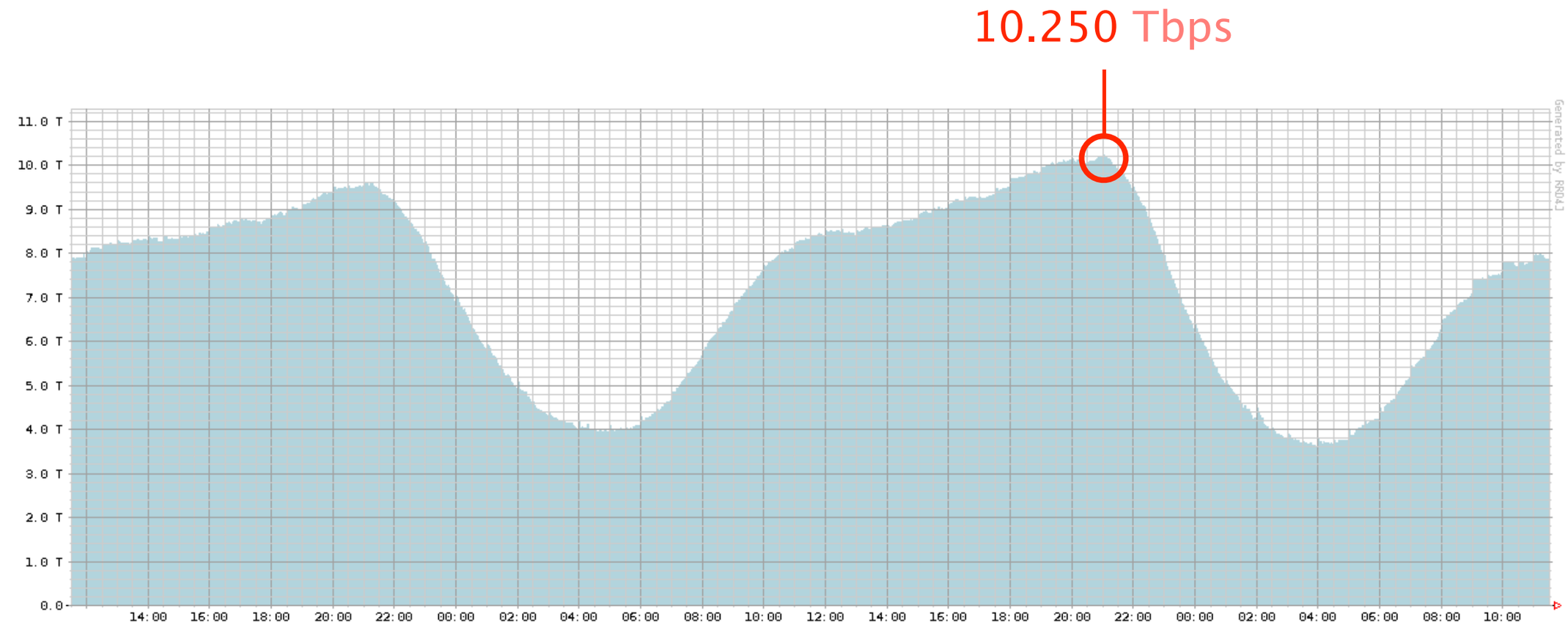


32x QSFP28 ports

25/40/50/100 GbE

8x Wedge100BF-32X. Total capacity: 25.6 Tbps

>2x what DE-CIX Frankfurt sees at peak time!



https://www.de-cix.net/en/locations/germany/frankfurt/statistics

Course Organization

This lecture is divided in two blocks

Lectures/Exercices

~8 weeks

Group project

~6 weeks

in teams of 3

Lectures/Exercices

~8 weeks

Group project

~6 weeks

in teams of 3

There will be 2h of lectures & 2h of hands-on exercises

Tue 14–16 Lecture

Tue 16–18 Practical, hands-on exercises

Exercises are *not* graded *but* will help for the project & exam

Lectures/Exercices

~8 weeks

Group project

~6 weeks

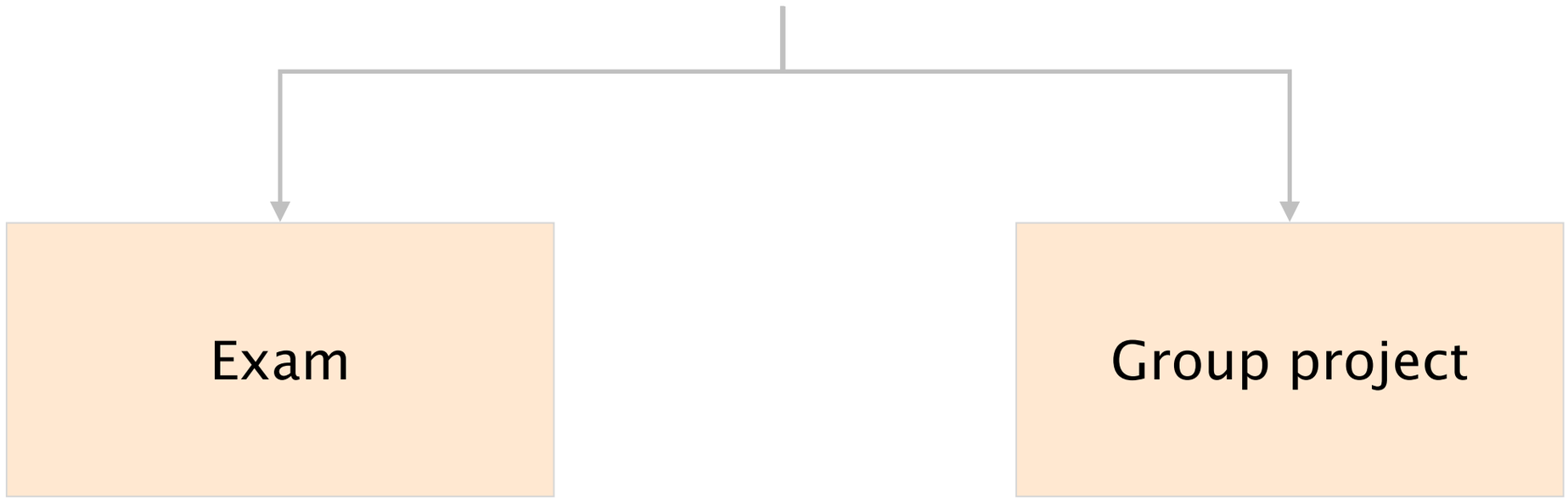
in teams of 3

In the project,
you'll partake in a class-wide challenge

Make ***your*** network ***the best***

We'll provide more information in the coming weeks

and, yes, *there will be prizes* for the best teams!



70%

written

30%

code, report, and presentation

Exam

70%

written

Examples (more in the lecture)

Implement a solution for <X> in P4

Optimize the network design for <X>

Is the P4 program <X> correct?

… so it's **crucial** to do the exercises

Your dream team of teaching assistants

Romain

coordinator

Edgar

Albert

Ege

Roland

+ Joël, Lukas

who followed the lecture last year

We'll use adv-net.ethz.ch as our official course platform

check it out regularly

Advanced Topics in Communication Networks

Fall 2022

Networked Systems
ETH Zürich — seit 2015

This course covers advanced topics and technologies in computer networks, both theoretically and practically. In the Fall 2022, the course will cover advanced topics in Internet routing and forwarding.

The goal for this course is to provide students with a deeper understanding of existing and upcoming Internet routing and forwarding technologies used in large-scale computer networks such as Internet Service Providers (e.g., Swisscom or Deutsche Telekom), Content Delivery Networks (e.g., Netflix) and Data Centers (e.g., Google).

Besides covering the fundamentals, the course will be “hands-on” and will enable students to play with the technologies in realistic network environments, and even implement some of them on their own during labs and a final group project.

News

Aug 30	We are getting ready for the upcoming semester, can't wait to kick off a new edition of AdvNet 🚀 If you intend to follow the course, you can already join the course's Slack workspace. It is necessary to use an @ethz.ch or @student.ethz.ch email address for signing up. If you do not have such an address, contact us.
Mar 22	Website for the Fall 2022 course edition goes live (yes, we are early). Please stay tuned for more contents. If you are interested in the materials from the previous years, you will find everything in the p4-learning repository on GitHub.

Contact

Location & time

We'll use Slack to discuss about
the course, exercises, and projects

Register today using your real name

see link on **https://adv-net.ethz.ch**

Should ***you*** take this course?

If you like computer networks, ***yes!***

If you like computer networks, ***yes!***

that said…

If you like computer networks, ***yes!***

that said…

You shouldn't take the course if

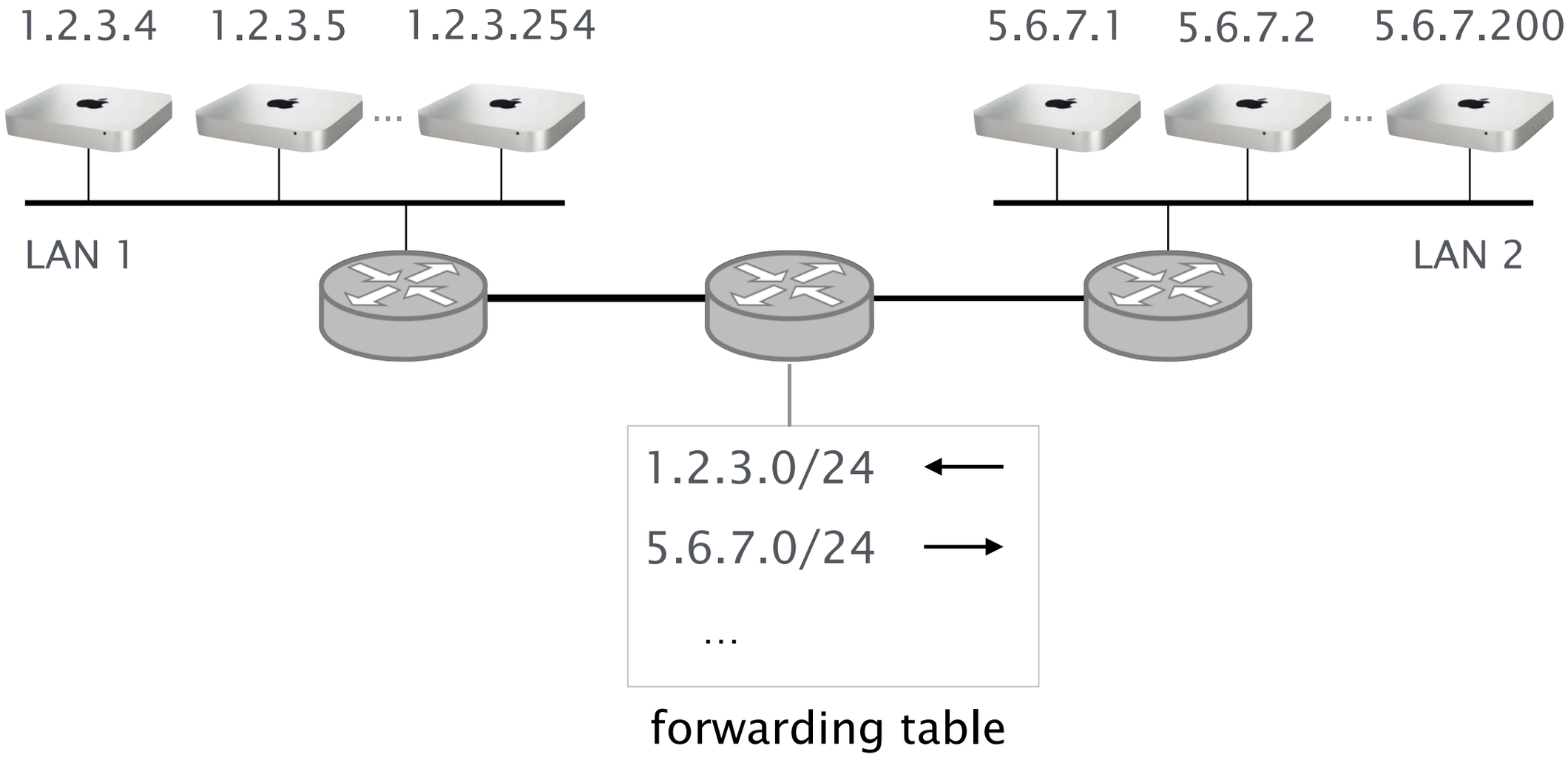
- you *hate* programming
- you can't work during the semester
- you expect 10+ years of exam history

Your first

program

IP forwarding

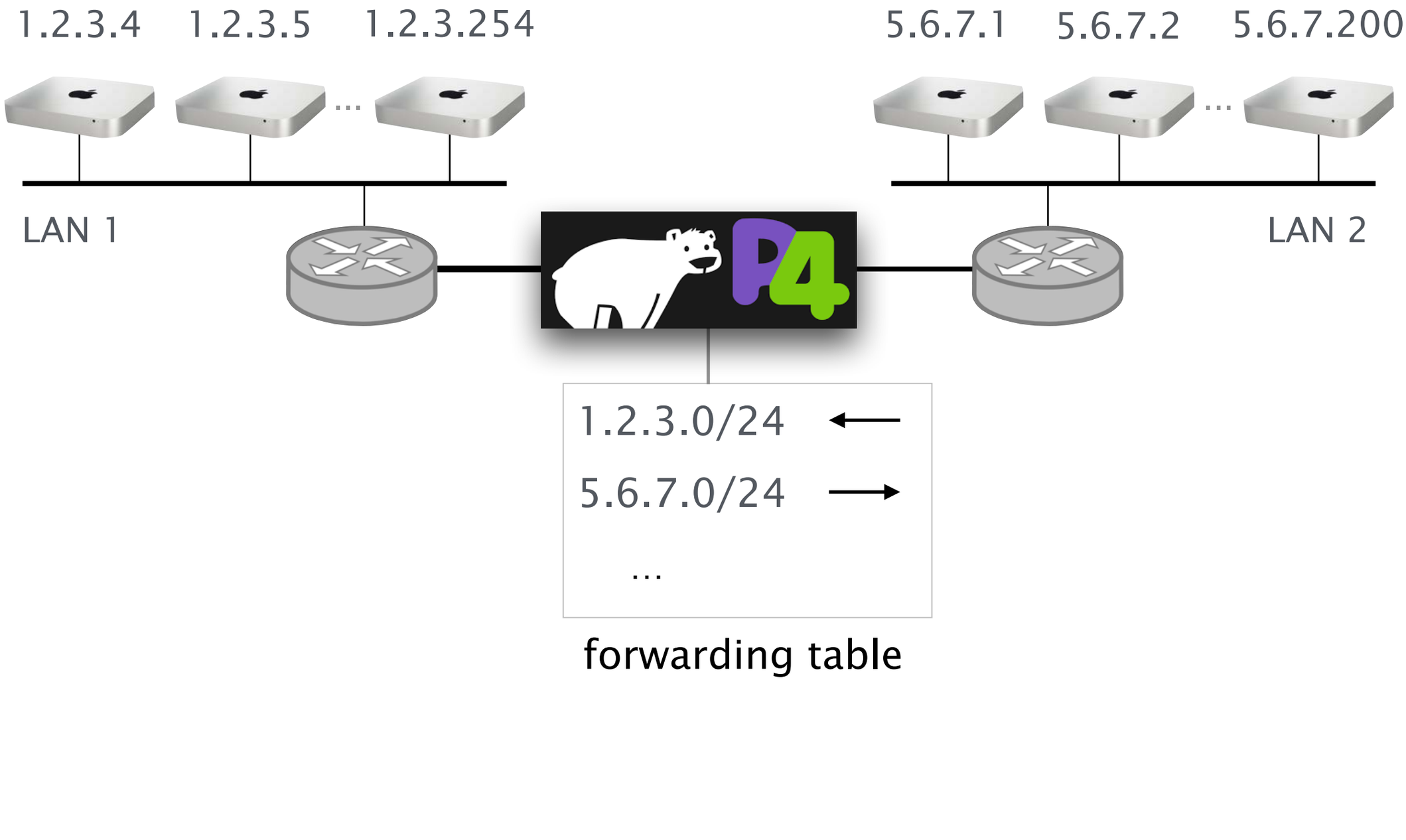
in a traditional router



forwarding table

IP forwarding

in P4



When forwarding an IP packet,

an IP router performs four actions:

- Lookup the next hop to use
- Update the MAC addresses
- Decrement the TTL
- Forward the packet to the output port

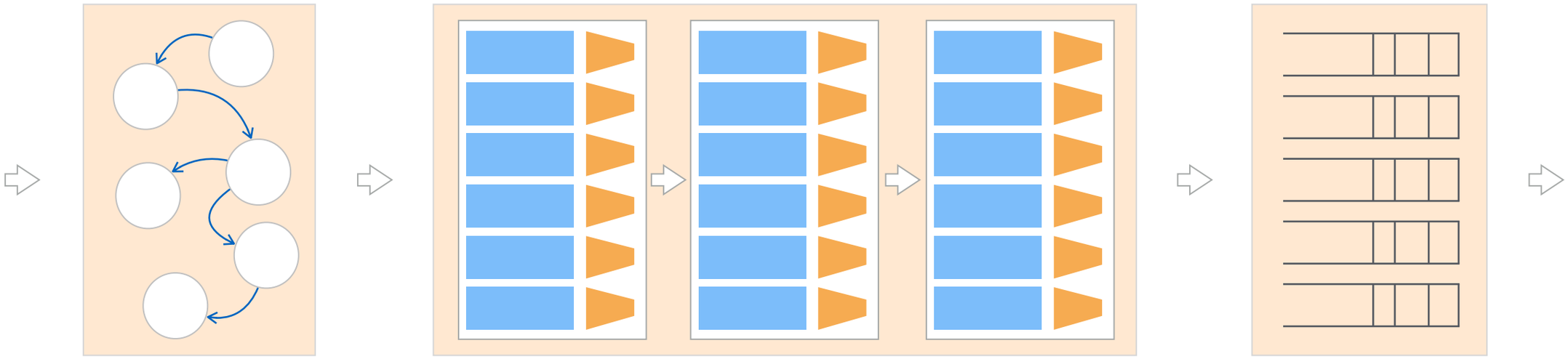
Each of them should be in implemented in P4

A P4 program consists of three basic parts

Parser

Match-Action Pipeline

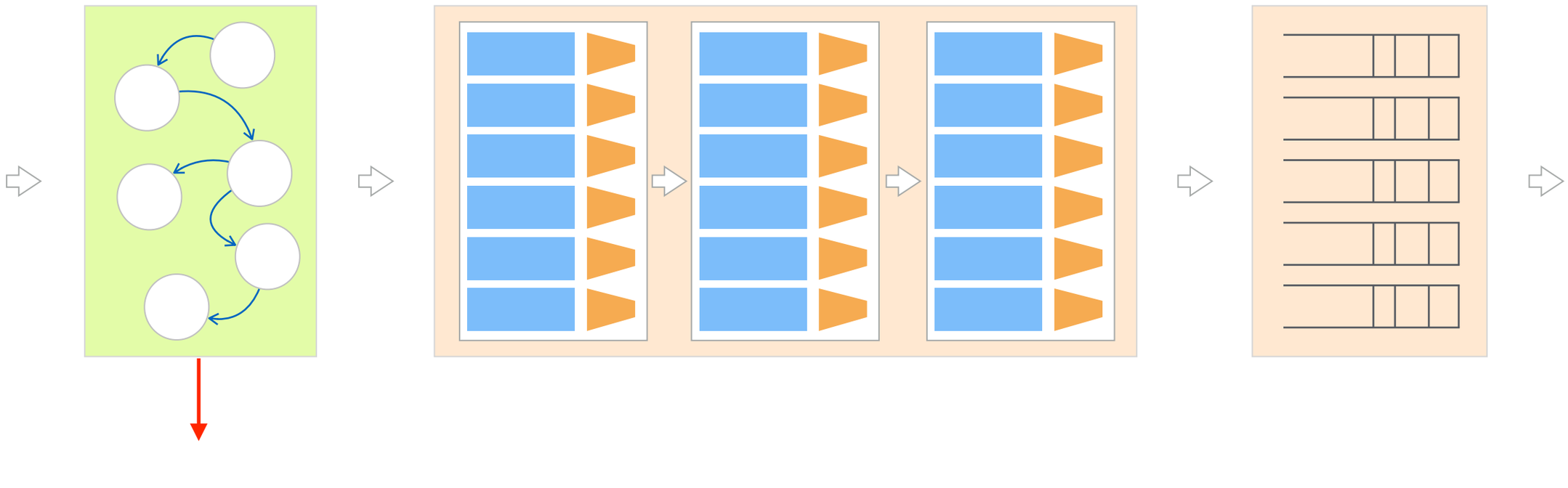
Deparser



Parser

Match-Action Pipeline

Deparser

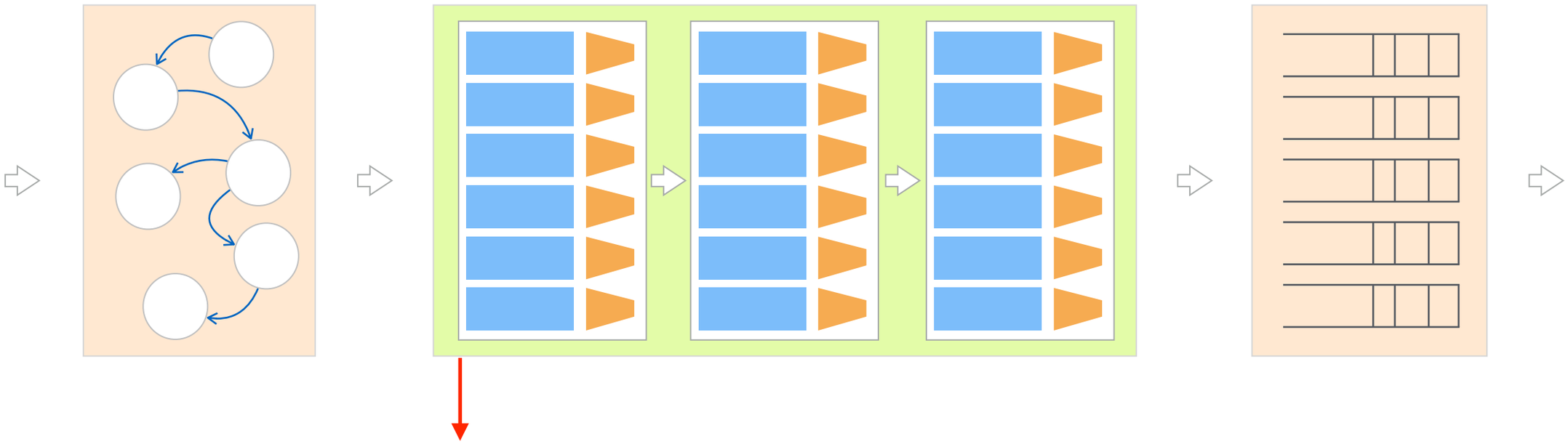


Specifies the headers to extract
from the incoming packet

Parser

Match-Action Pipeline

Deparser

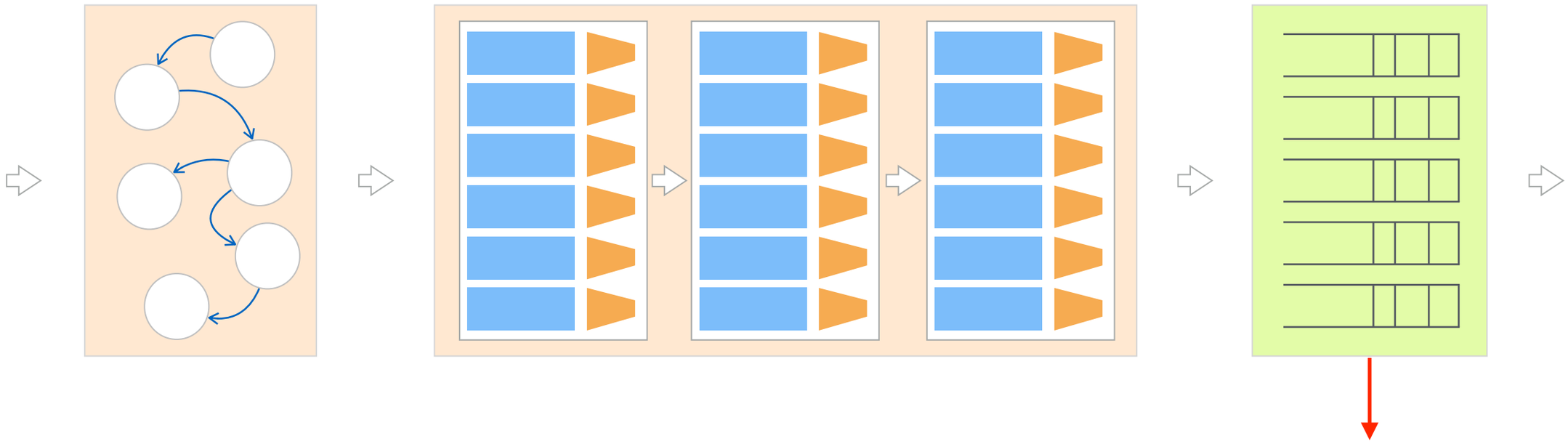


Specifies the processing logic to apply on the extracted headers

Parser

Match-Action Pipeline

Deparser

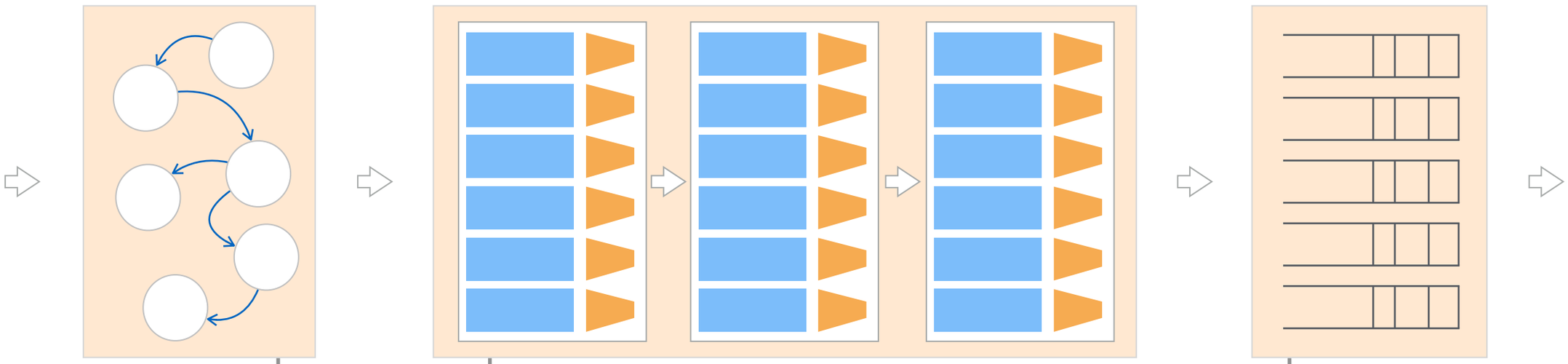


Specifies how
the output packet
will look on the wire

Parser

Match-Action Pipeline

Deparser



```
V1Switch(
 MyParser(),
 MyVerifyChecksum(),
 MyIngress(),
 MyEgress(),
 MyComputeChecksum(),
 MyDeparser()
) main;
```

```
#include <core.p4>
#include <v1model.p4>
```

Libraries

```
const bit<16> TYPE_IPV4 = 0x800;
typedef bit<32> ip4Addr_t;
header ipv4_t {…}
struct headers {…}
```

Declarations

```
parser MyParser(…) {
    state start {…}
    state parse_ethernet {…}
    state parse_ipv4 {…}
}
```

Headers extraction

```
control MyIngress(…) {

    action ipv4_forward(…) {…}

    table ipv4_lpm {…}

    apply {
        if (…) {…}
    }
}
```

Processing logic

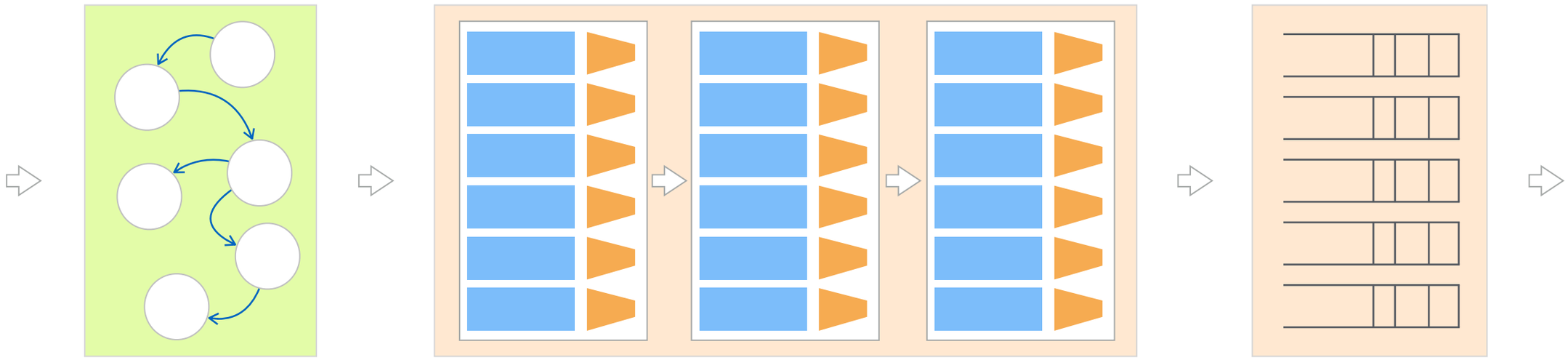
```
control MyDeparser(…) {…}
```

Output packet assembly

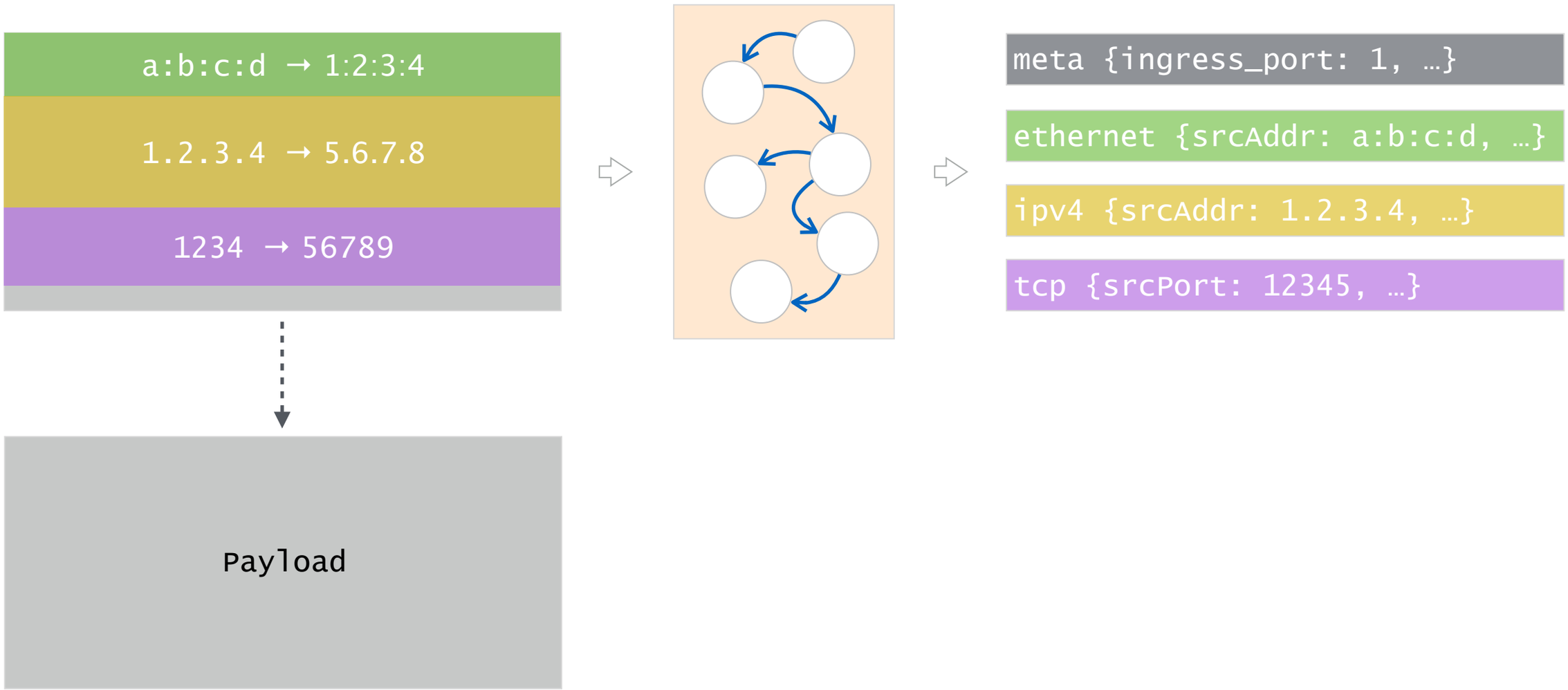
```
V1Switch(
 MyParser(),
 MyVerifyChecksum(),
 MyIngress(),
 MyEgress(),
 MyComputeChecksum(),
 MyDeparser()
) main;
```

“main()”

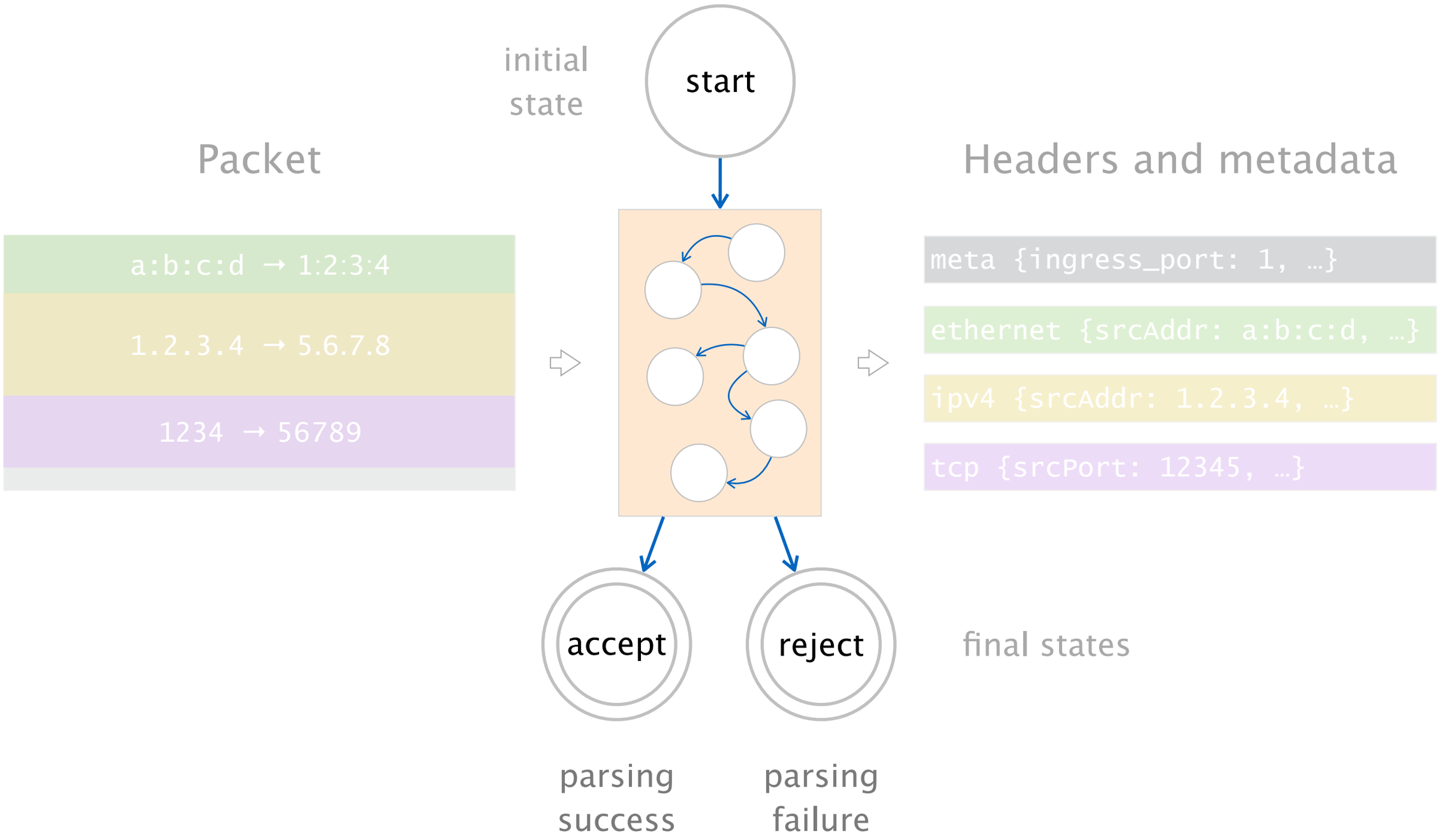
Parser

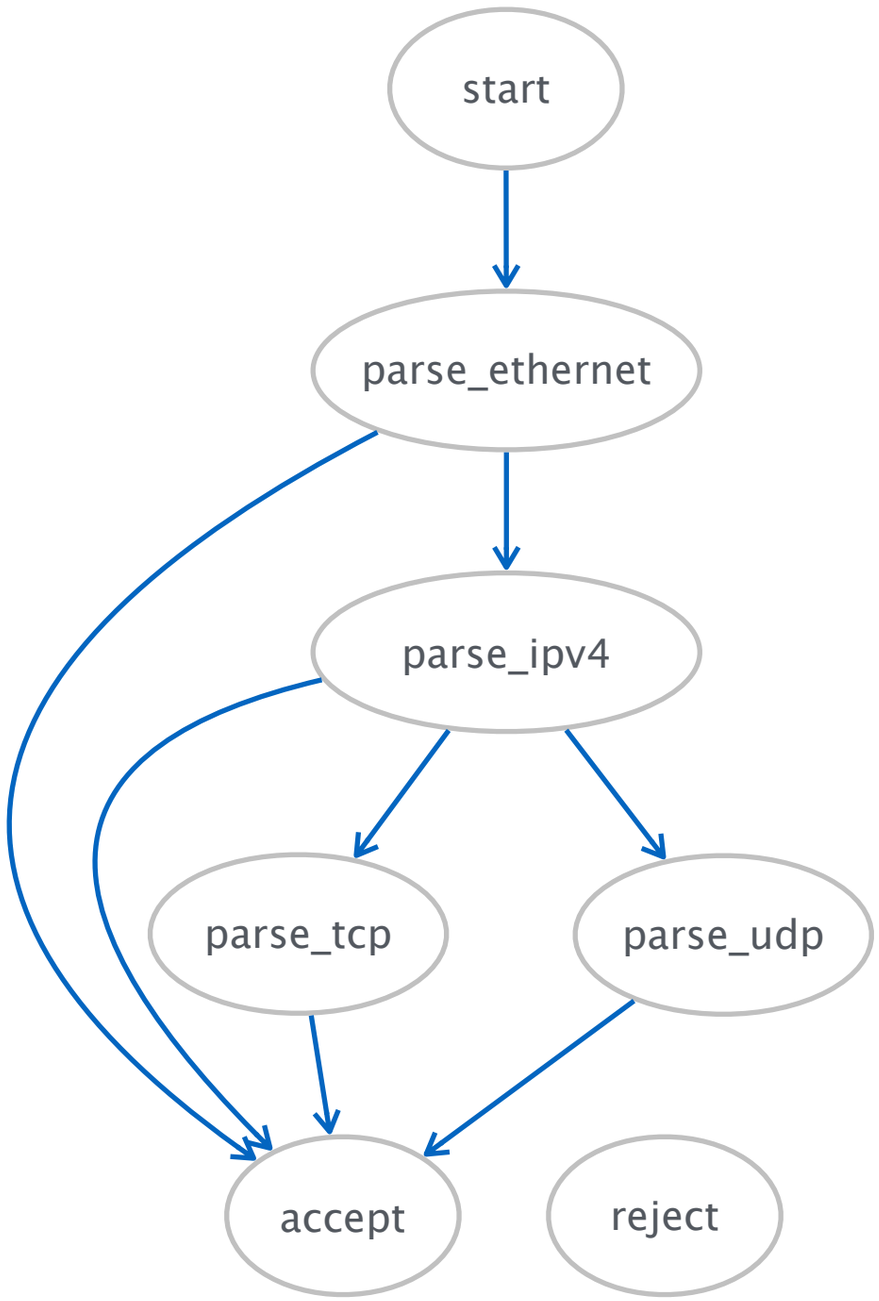


The parser relies on a programmable state machine which maps packets into headers and metadata



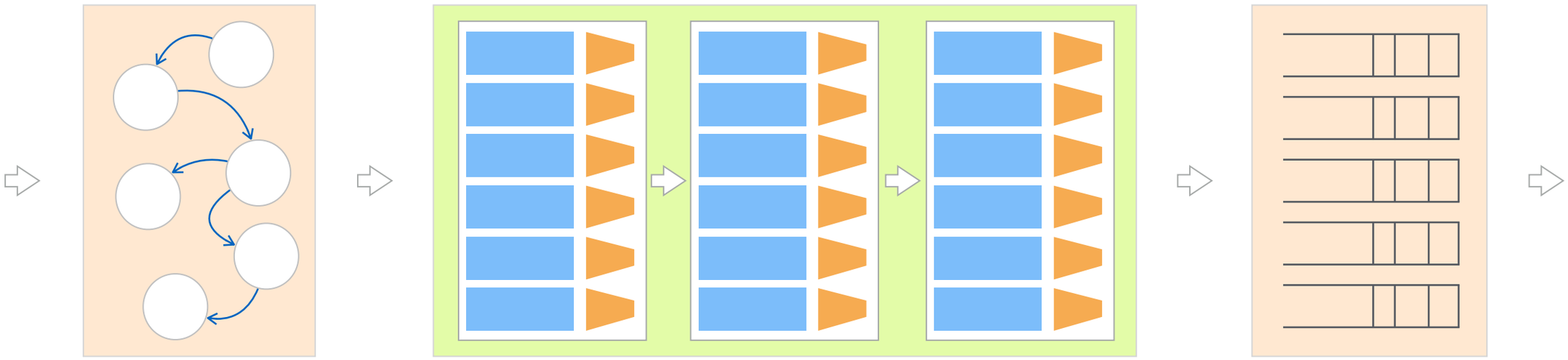
The state machine has three predefined states: start, accept and reject






```
parser MyParser(…) {
 state start {
  transition parse_ethernet;
 }
 state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
 }
 state parse_ipv4 {
  packet.extract(hdr.ipv4);
  transition select(hdr.ipv4.protocol) {
   6: parse_tcp;
   17: parse_udp;
   default: accept;
  }
 }
 state parse_tcp {
  packet.extract(hdr.tcp);
   transition accept;
 }
 state parse_udp {
  packet.extract(hdr.udp);
   transition accept;
 }
}
```

Match-Action Pipeline



In the match-action pipeline,
control blocks manipulate headers and metadata

A control block is an imperative program which describes how to process packets

A control block is an imperative program which describes how to process packets

Headers and metadata from parser

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  bit<9> port;

  apply {
    port = 1
    std_meta.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = 0x2;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
}
```

Variable declaration

Control flow

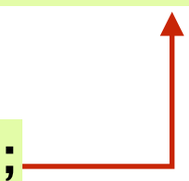
Actions allow to re-use code

in a block or globally, they are similar to C functions

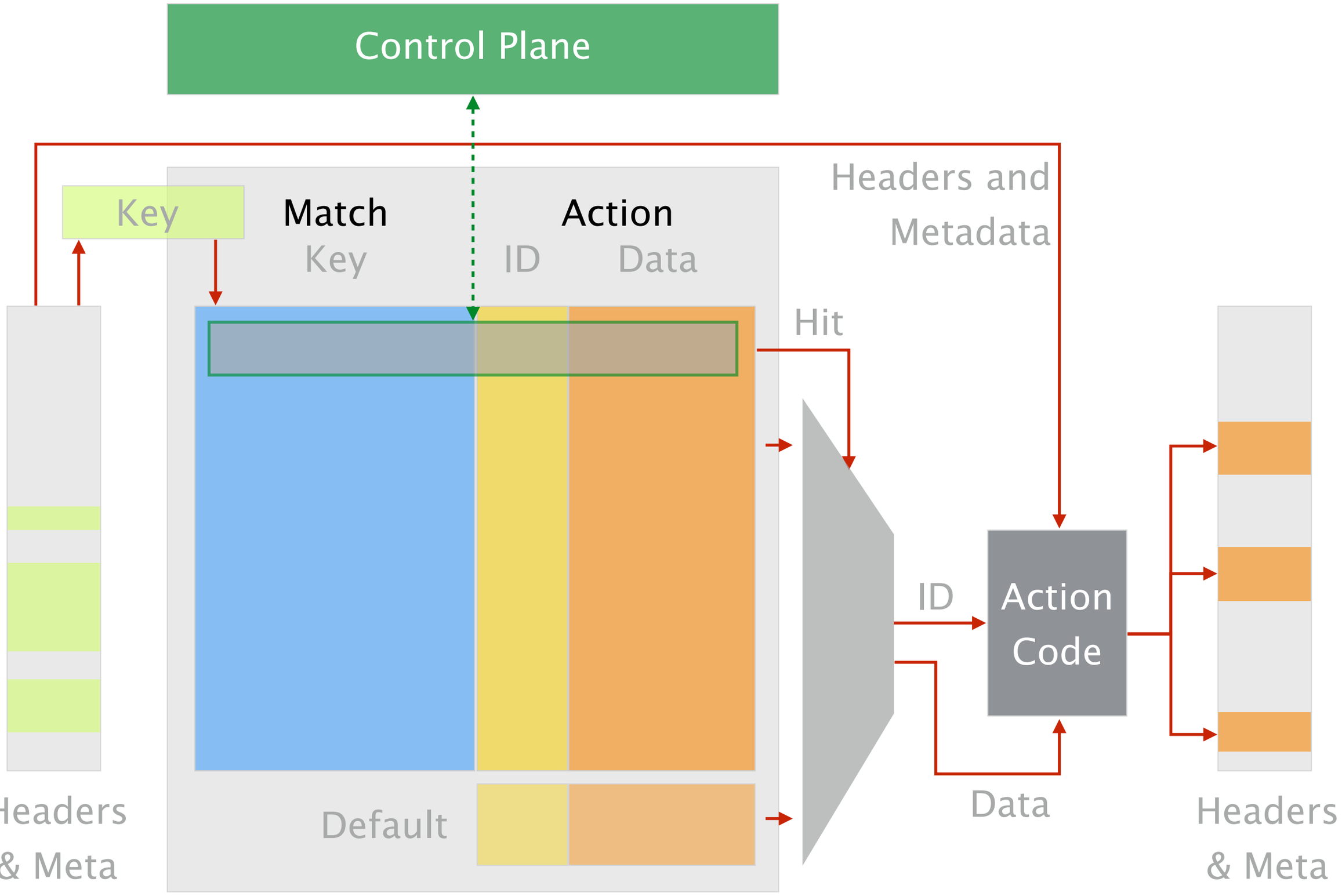
```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

 action ipv4_forward(macAddr_t dstAddr,
                     egressSpec_t port) {
  std_meta.egress_spec = port;
  hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
  hdr.ethernet.dstAddr = dstAddr;
  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
 }

 apply {
  ipv4_forward(0x123, 1);
 }
}
```



Control blocks often rely on tables which map specific headers values to specific actions


Control Plane
Headers and
Metadata
Key
Match
Key
Action
ID
Data
Hit
Default
ID
Action
Code
Data
Headers
& Meta
Headers
& Meta

```
table                    {
 key = {
                         :    ;
 }
 actions = {
 }

 size =     ;

 default_action =     ;
}
```

Table name

Field(s) to match

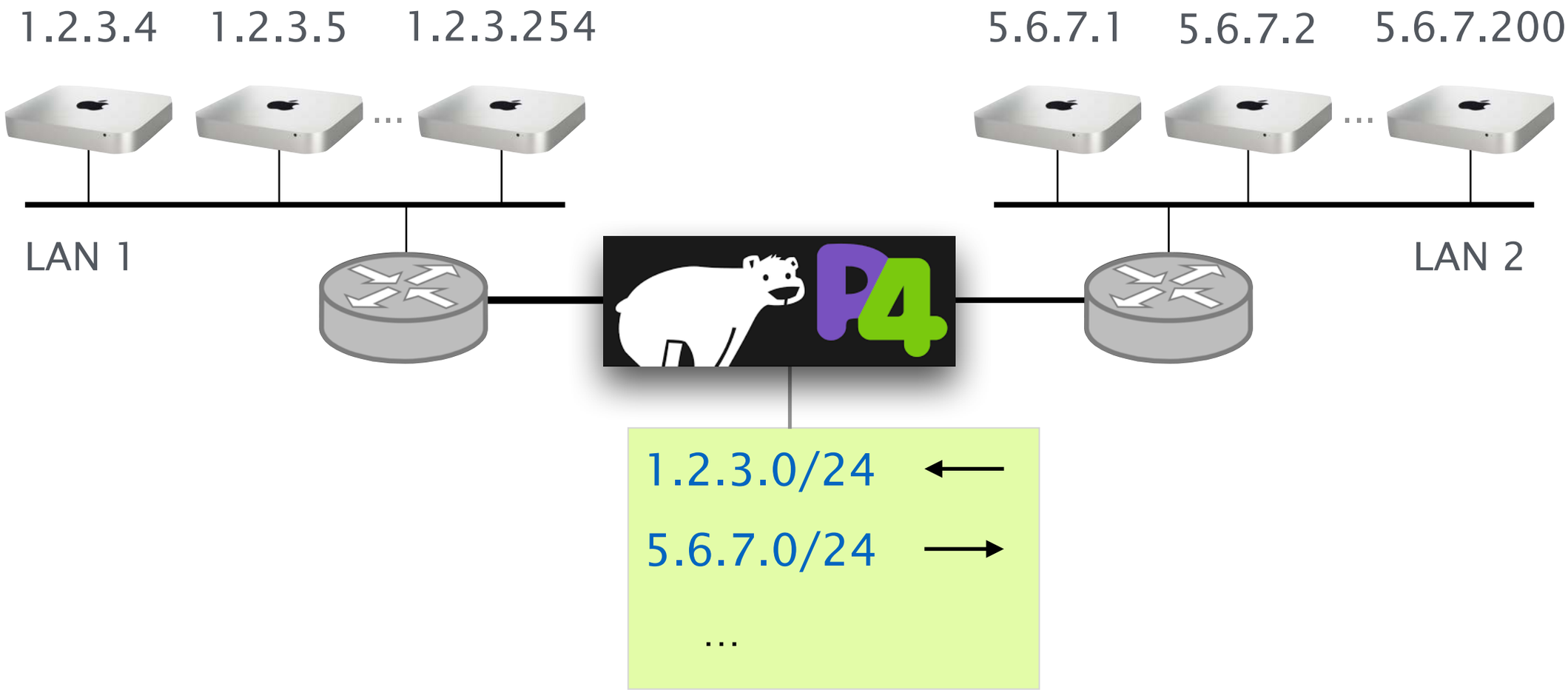
Match type

Possible actions

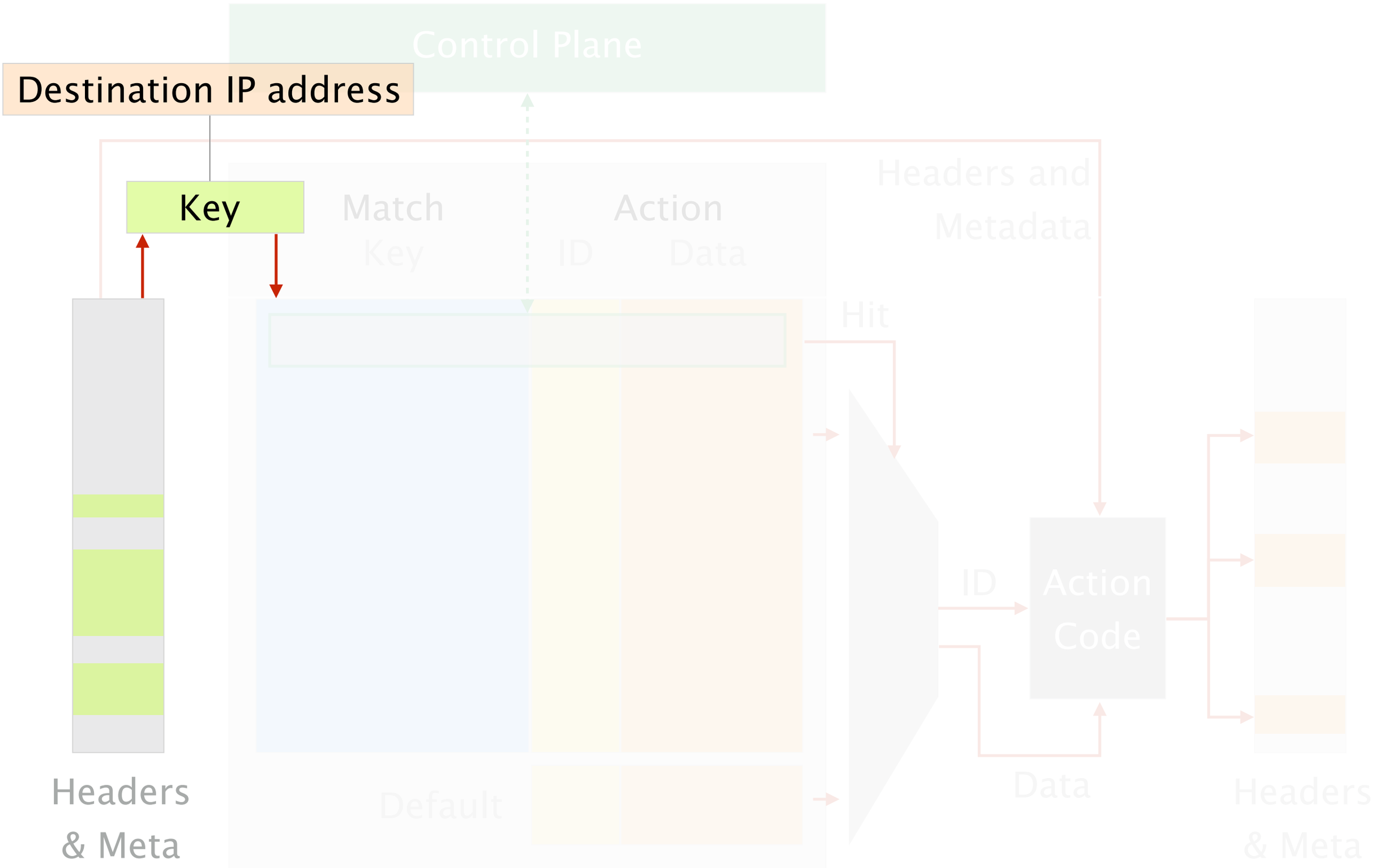
Max. # entries in table

Default action

Example: IP forwarding table



forwarding table


Control Plane
Destination IP address
Key
Match
Key
Action
ID
Data
Headers and
Metadata
Hit
Default
ID
Action
Code
Data
Headers
& Meta
Headers
& Meta

Control Plane

Longest prefix match

Match
Key

Key

Action
ID Data

Headers and
Metadata

Hit

ID

Action
Code

Data

Headers
& Meta

Default

Headers
& Meta

1: ipv4_forward(mac,port)
2: drop()

Action

ID Data

Key

Match Key

Headers and Metadata

Hit

ID

Action Code

Data

Headers & Meta

Default

Headers & Meta

```
table ipv4_lpm {
 key = {
  hdr.ipv4.dstAddr: lpm;
 }
 actions = {
  ipv4_forward;
  drop;
 }

 size = 1024;

 default_action = drop();
}
```

Table name

Destination IP address

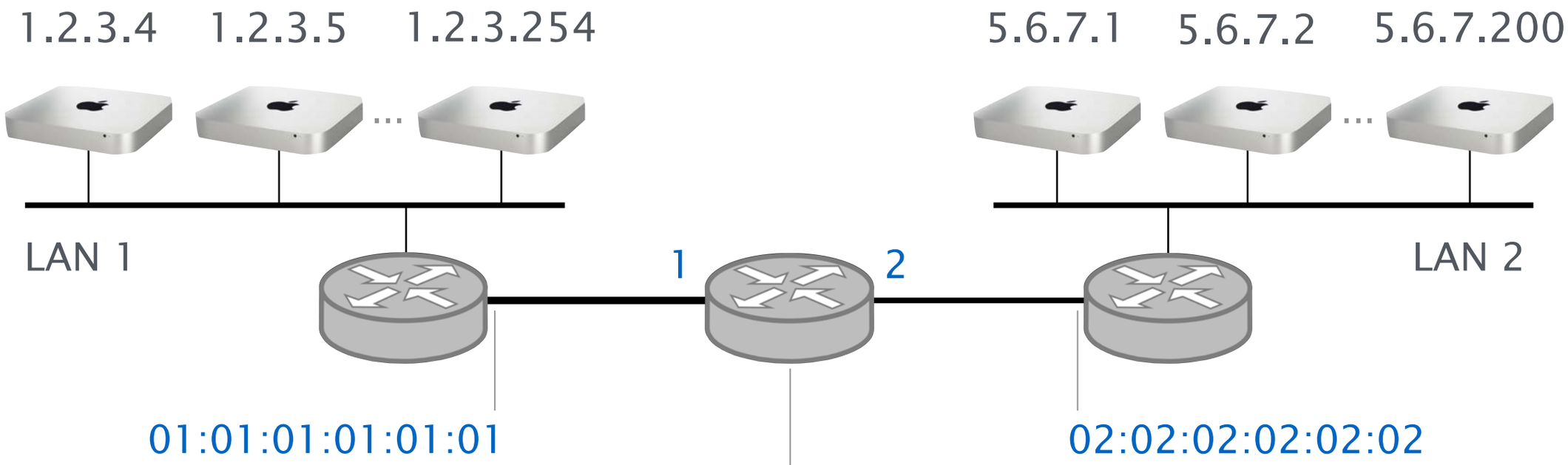
Longest prefix match

Possible actions

Max. # entries in table

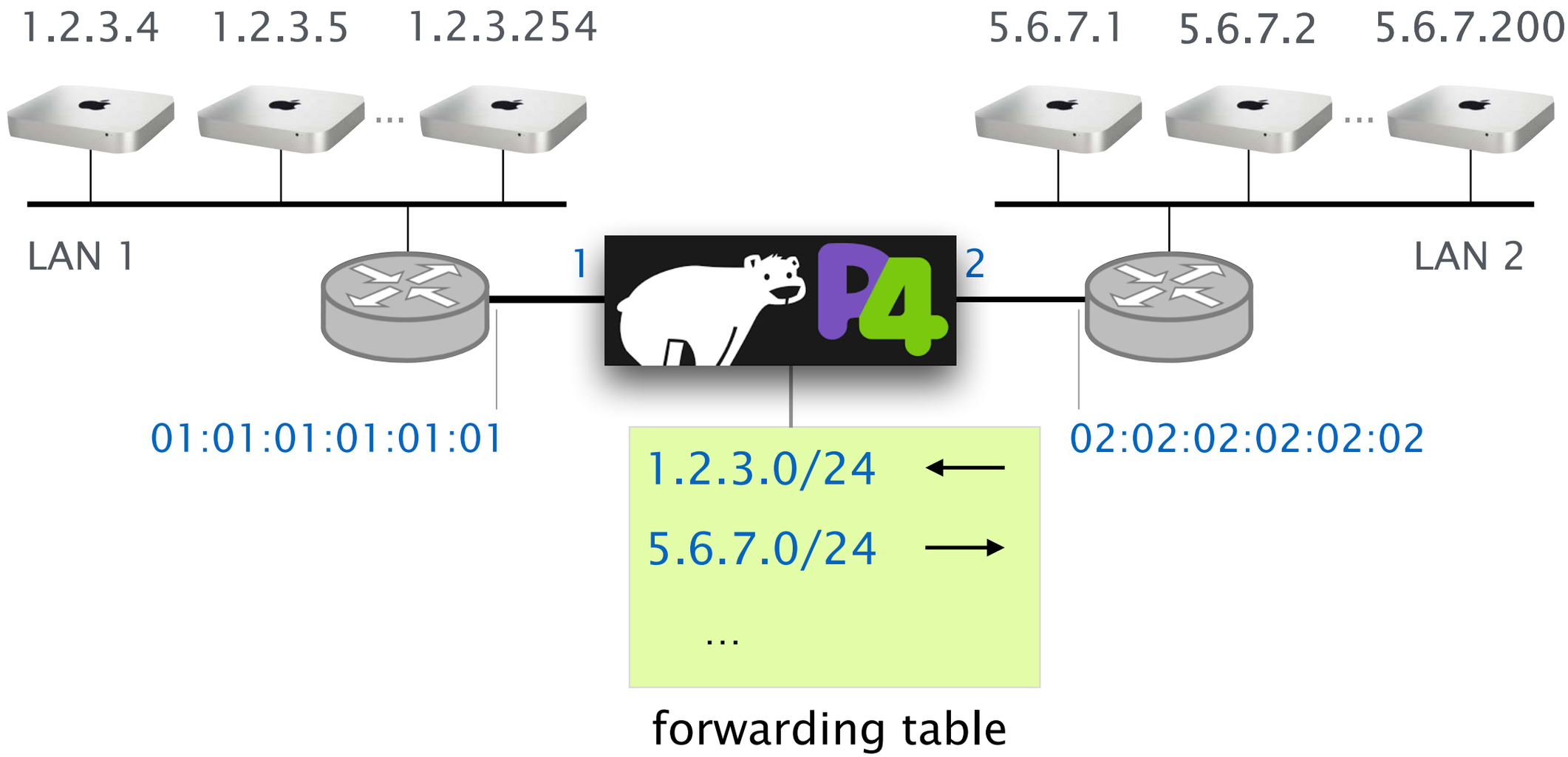
Default action

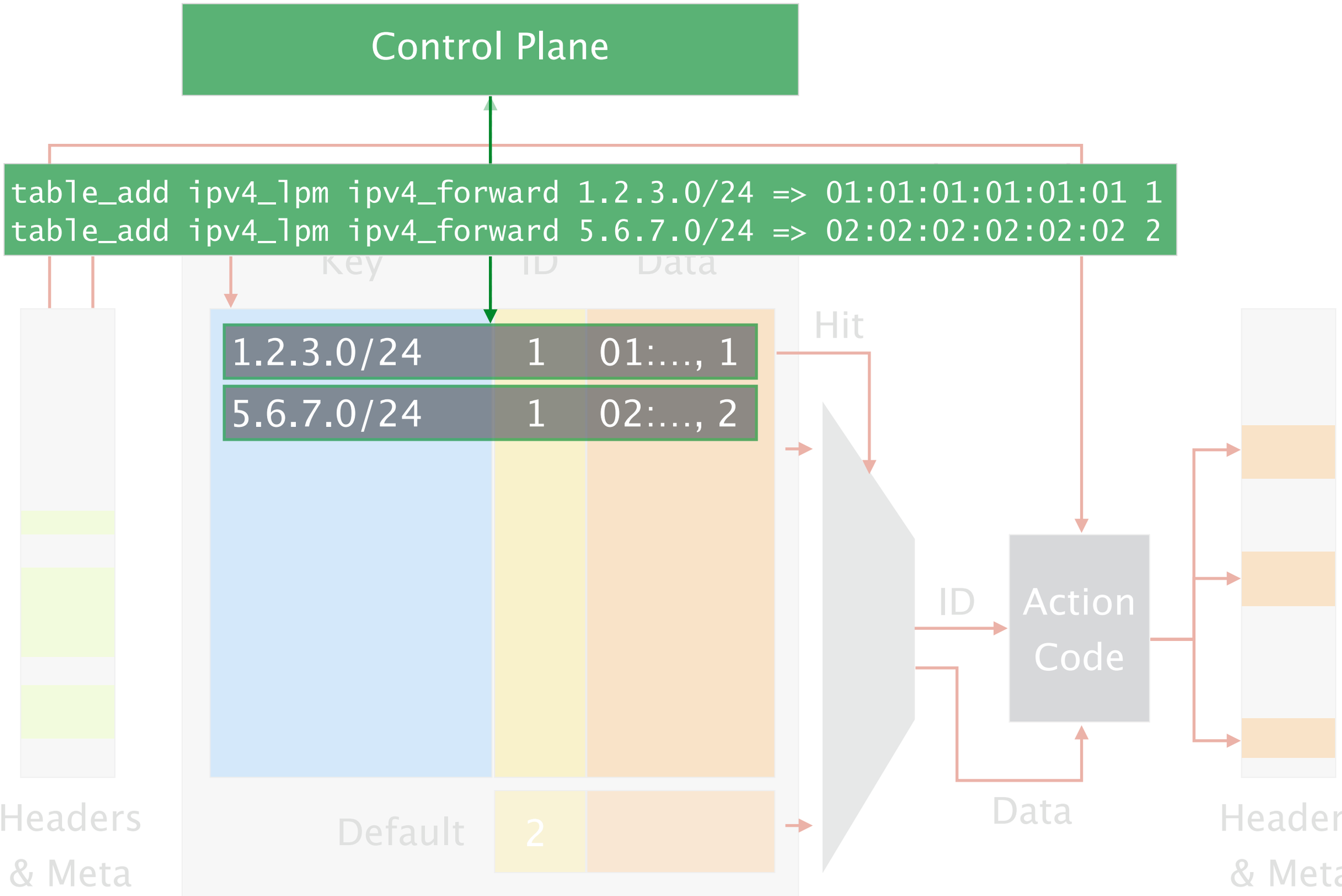
Example: IP forwarding table



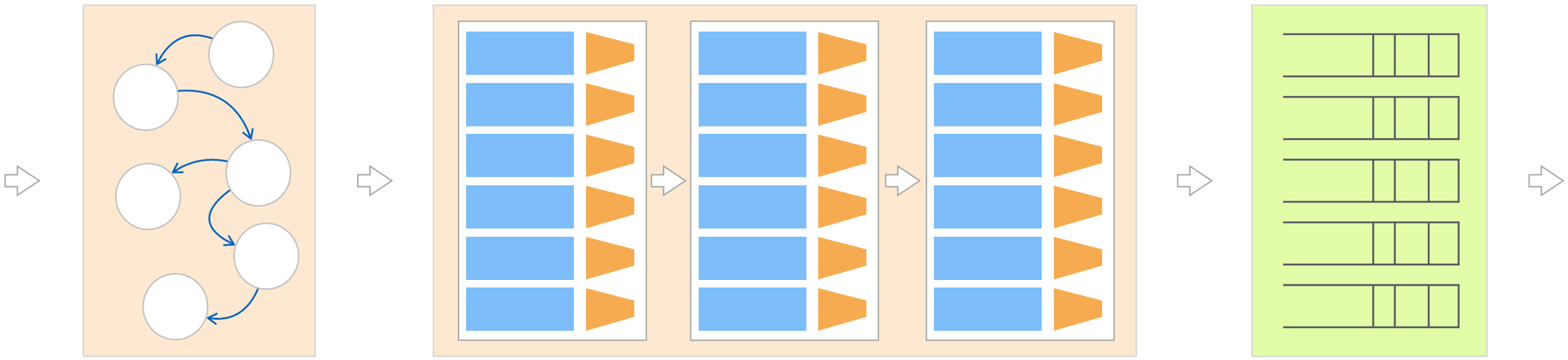

```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    standard_metadata.egress_spec = port;
}
```

Example: IP forwarding table

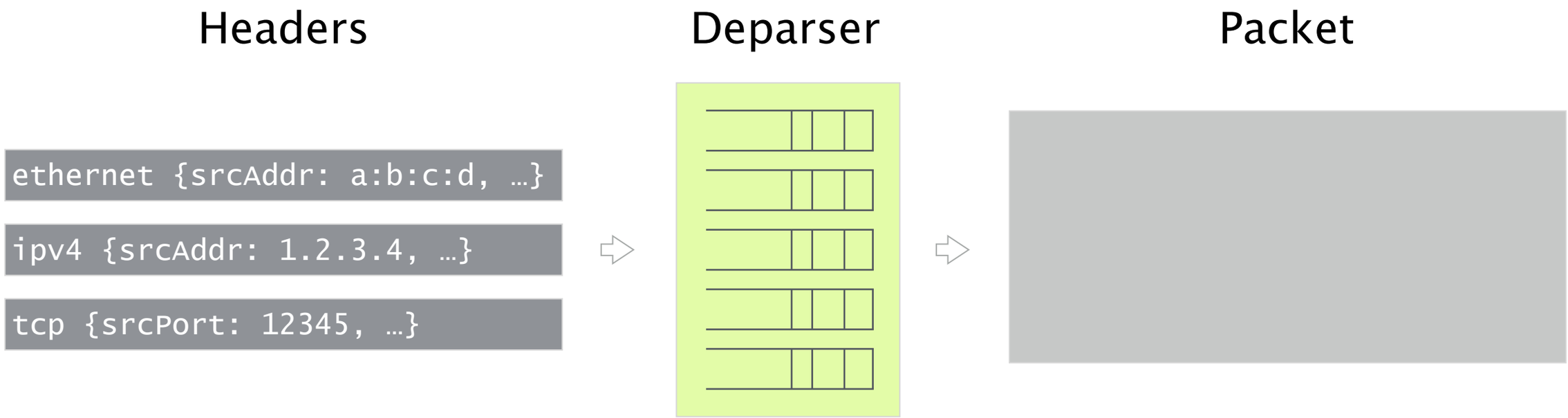



Control Plane
table_add ipv4_lpm ipv4_forward 1.2.3.0/24 => 01:01:01:01:01:01 1
table_add ipv4_lpm ipv4_forward 5.6.7.0/24 => 02:02:02:02:02:02 2
Key
ID
Data
1.2.3.0/24
1
01:..., 1
5.6.7.0/24
1
02:..., 2
Default
2
Hit
ID
Action Code
Data
Headers & Meta
Headers & Meta

Deparser



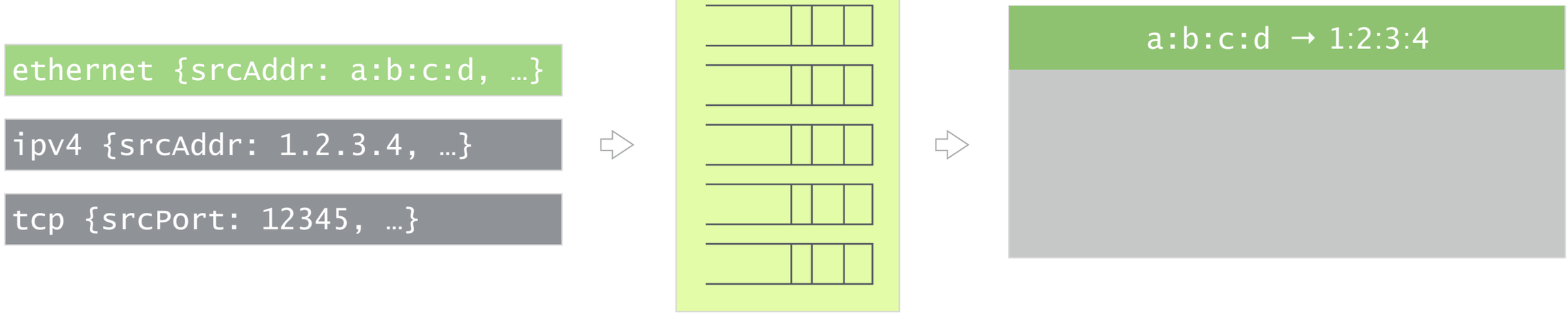
The Deparser assembles the headers back into a well-formed packet



Headers

Deparser

Packet



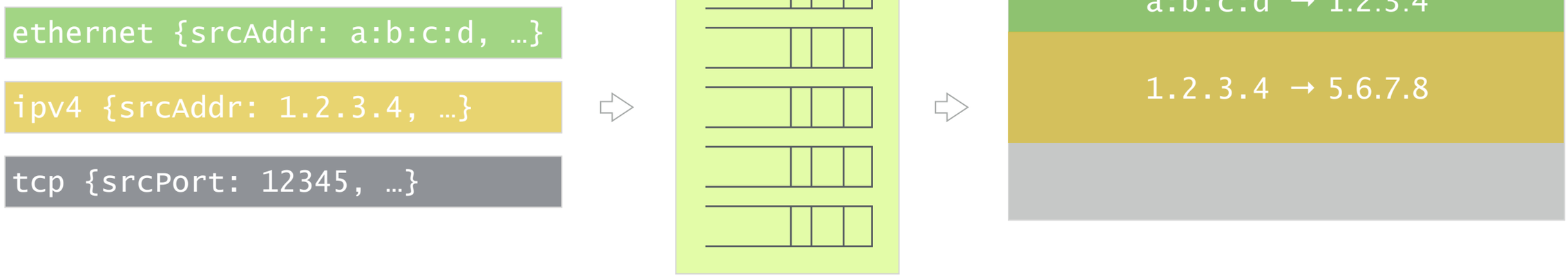

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);


    }
}
```

Headers

Deparser

Packet



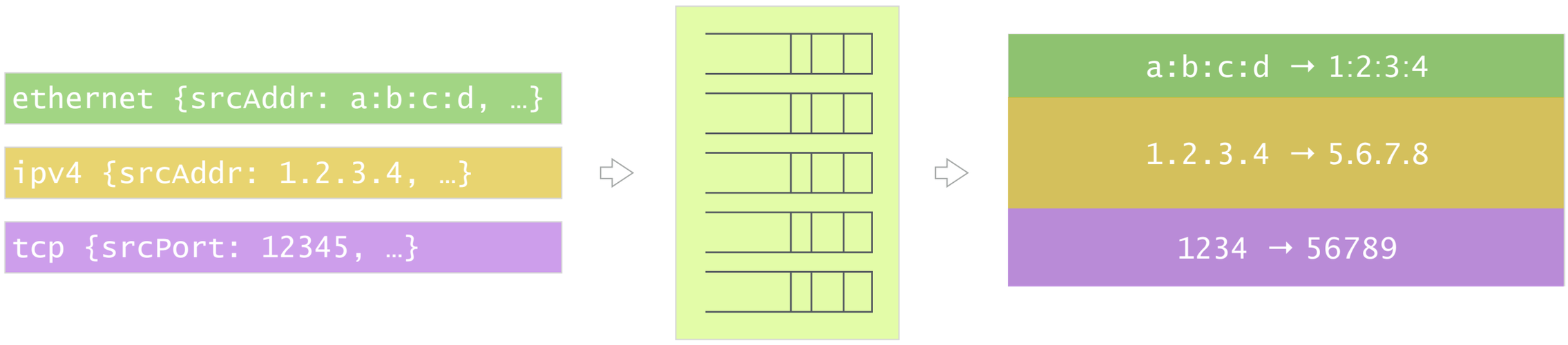

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);

    }
}
```

Headers

Deparser

Packet




```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.tcp);
    }
}
```

Headers

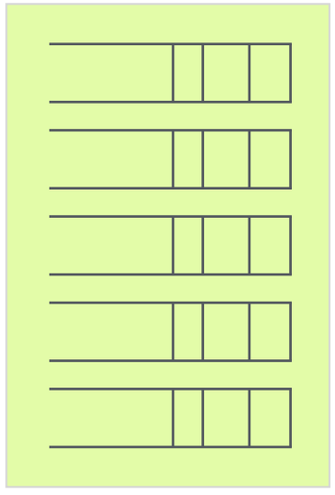
Deparser

Packet

ethernet {srcAddr: a:b:c:d, …}

ipv4 {srcAddr: 1.2.3.4, …}

tcp {srcPort: 12345, …}



a:b:c:d → 1:2:3:4

1.2.3.4 → 5.6.7.8

1234 → 56789

Payload

A quick look at how *we* use

in our research

We used P4 to…

Defend against
next-gen DDoS attacks
that morph every few secs

ACM SIGCOMM 2022

Aggregate-Based Congestion Control for Pulse-Wave DDoS Defense

Albert Gran Alcoz
ETH Zürich

Martin Strohmeier
Armasuisse

Vincent Lenders
Armasuisse

Laurent Vanbever
ETH Zürich

ABSTRACT

Pulse-wave DDoS attacks are a new type of volumetric attack formed by short, high-rate traffic pulses. Such attacks target the Achilles' heel of state-of-the-art DDoS defenses: their reaction time. By continuously adapting their attack vectors, pulse-wave attacks manage to render existing defenses ineffective.

In this paper, we leverage programmable switches to build an in-network DDoS defense effective against pulse-wave attacks. To do so, we revisit Aggregate-based Congestion Control (ACC): a mechanism proposed two decades ago to manage congestion events caused by high-bandwidth traffic aggregates. While ACC proved efficient in inferring and controlling DDoS attacks, it cannot keep up with the speed requirements of pulse-wave attacks.

We propose *ACC-Turbo*, a renewed version of ACC that infers attack patterns by applying online-clustering techniques in the network and mitigates them by leveraging programmable packet scheduling. By doing so, *ACC-Turbo* identifies attacks at line rate and in real-time, and rate-limits attack traffic on a per-packet basis.

We fully implement *ACC-Turbo* in P4 and evaluate it on a wide range of attack scenarios. Our evaluation shows that *ACC-Turbo* autonomously identifies DDoS attack vectors in an unsupervised manner and rapidly mitigates pulse-wave DDoS attacks. We also show that *ACC-Turbo* runs on existing hardware (Intel Tofino).

CCS CONCEPTS

• **Networks** → **Network security**; **Denial-of-service attacks**;

KEYWORDS

Network Security, DDoS, Pulse-Wave DDoS, ACC, Aggregate-Based Congestion Control, Programmable Scheduling, Network Defenses

ACM Reference Format:
Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. 2022. Aggregate-Based Congestion Control for Pulse-Wave DDoS Defense. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3544216.3544263

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544263

1 INTRODUCTION

Pulse-wave DDoS attacks have recently managed to take down critical network infrastructure while causing enormous financial and reputational damages [2, 4, 49, 57]. In contrast to conventional DDoS attacks, which grow steadily and persist longer in time, pulse-wave DDoS attacks consist of high-rate short-lived bursts. Each burst typically leverages a different attack vector (e.g., NTP, DNS, Memcached) and can reach hundreds of Gbps [38, 50, 52].

The threat in pulse-wave attacks resides in that they target the Achilles' heel of existing DDoS mitigation systems: their reaction time. Most in-network DDoS defenses today (both in research and production) rely on some sort of offline facility that can either directly scrub traffic [7, 19, 23, 34, 42, 58], orchestrate a routing-based defense [46, 51], or deploy an in-network pre-configured mitigation [35, 54]. The time required for an in-network defense to reach this external facility and deploy the corresponding defense can be in the order of seconds to minutes [52]. Pulse-wave attacks exploit this vulnerability by sending traffic pulses that force the DDoS defense to repeat this control loop over and over. By keeping the defenses in a constant transient state, pulse-wave attacks manage to make them ineffective. If the defenses rely on traffic redirection, pulse-wave attacks may even produce route flapping in the network [26].

Designing a pulse-wave defense is intricate. Like conventional-DDoS defenses, an ideal pulse-wave defense needs to be, first, *generic*, to identify a wide variety of attack vectors at different granularity [5, 17]. Generic techniques usually require unsupervision and incur the risk of misclassifying traffic. Thus, an ideal defense also needs to be *measured* in responding to attacks [47]. Most DDoS defenses fail at the first condition. For example, signature-based defenses [35, 54, 58] are not generic. They only cover a small set of attack vectors, and can not keep up with the constantly-growing list of new attacks [5, 17]. Similarly, most congestion-management tools (e.g., heavy-hitter detectors or active queue management) lack granularity: they only work at, e.g., the per-flow level or the whole-traffic level. Other DDoS defenses fail at satisfying the second condition. For example, drop- or routing-based defenses [20, 39] strongly degrade performance in case of misclassification.

Aggregate-Based Congestion Control (ACC) was proposed two decades ago, *satisfying the two* design conditions [36]. ACC is a canonical mechanism to reduce the impact of congestion caused by *generic* traffic aggregates, with a *measured* bandwidth control. At a high level, ACC is a feedback loop that iteratively: *(i)* infers the aggregates causing the congestion; before *(ii)* reducing their throughput to a reasonable level. To infer the aggregates, ACC clusters the headers of packets dropped by a Random Early Detection (RED) queuing discipline. ACC then rate-limits the inferred aggregates to keep the total traffic throughput below the link capacity.

We used P4 to…

Detect runtime errors that drop arbitrary packets

possibly nondeterministically

ACM SIGCOMM 2022

FAst In-Network *GraY* Failure Detection for ISPs

Edgar Costa Molero
ETH Zurich
cedgar@ethz.ch

Stefano Vissicchio
University College London
s.vissicchio@ucl.ac.uk

Laurent Vanbever
ETH Zurich
lvanbever@ethz.ch

ABSTRACT

Avoiding packet loss is crucial for ISPs. Unfortunately, malfunctioning hardware at ISPs can cause long-lasting packet drops, also known as gray failures, which are undetectable by existing monitoring tools.

In this paper, we describe the design and implementation of FANcY, an ISP-targeted system that detects and localizes gray failures quickly and accurately. FANcY complements previous monitoring approaches, which are mainly tailored for low-delay networks such as data center networks and do not work at ISP scale. We experimentally confirm FANcY's capability to accurately detect gray failures in seconds, as long as only tiny fractions of traffic experience losses. We also implement FANcY in an Intel Tofino switch, demonstrating how it enables fine-grained fast rerouting.

CCS CONCEPTS

• **Networks** → **Network measurement**; Network simulations; **Programmable networks**; *In-network processing*;

KEYWORDS

Failure detection, Measurements, Network Hardware, Programmable data planes

ACM Reference Format:
Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. 2022. FAst In-Network *GraY* Failure Detection for ISPs. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3544216.3544242

1 INTRODUCTION

Avoiding packet loss is so critical to ISPs that research and industry efforts focus increasingly on ensuring minimal downtime upon failures (e.g., [26, 32, 39]). A major result of past efforts is that hard failures affecting all packets crossing a link or node are typically detected and dealt with quickly thanks to the BFD protocol [28].

In practice, however, malfunctioning hardware often causes packet losses only for subsets of packets sent over a link. Table 1 (further discussed in §2.2) shows representative examples of device bugs in this category. Additional examples include misplaced line cards and bent or dirty fibers [46].

In this paper, we call *gray failure* any hardware malfunction that causes non-transient packet loss on a subset of the traffic forwarded by any packet-forwarding device – which we generally call a switch.

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544242

Consistent with our definition, we do not classify congestion as a gray failure.

As confirmed by a survey we conducted (see §2.1), ISP operators consider gray failures a major concern, and lack techniques to detect and locate them. Indeed, they often become aware of gray failures only when customers complain about the failure-induced packet loss, which they end up troubleshooting for days or weeks.

The reason why existing techniques are ineffective is that detecting and localizing gray failures requires analyzing *all* the traffic. Hello protocols such as BFD do not work because most gray failures do not impact messages from these protocols. ISP monitoring tools, such as NetFlow [22] or sFlow [36], do not help either: since they rely on random packet sampling for scalability, they are unable to support fine-grained traffic analyses (as also shown in [41]), which would be needed to spot gray failures. Finally, mechanisms internal to switches, such as deflection on drop [44], do not capture several failure cases, including those where the drop flag is not correctly set on packets because of memory corruption, and link-level failures.

Of course, gray failures are not specific to ISPs, and recent contributions have proposed gray failure detectors for data center and cloud networks. Those detectors' designs, however, do not match the peculiarities of ISP networks: they either require control of end hosts [17, 23, 37, 40], assume low packet loss rates and extremely high-speed interfaces between the control and data planes (e.g., [31]), or require limited links delay and traffic volumes (e.g., [44]). We review the unsuitability of recent related work and simple system designs in §2.

Vision. We aim at designing an accurate and fast gray failure detector for ISPs. Similar to BFD, the immediate application of such a detector would be to support *selective fast rerouting on gray failures* – i.e., rerouting traffic only for the disrupted traffic, as fast as possible. We also envision that in the future, a gray failure detector may assist operators in finding the root cause of gray failures, and enable new control- and data-plane applications, such as automated failure repair through ad-hoc forward error correction mechanisms or hardware reconfiguration (e.g., [43]).

Problem statement. We focus on the following question:

> *Can we build an ISP-targeted system able to detect and localize intra-domain gray failures in seconds?*

By localizing we mean identifying both the switch port suffering from a gray failure and the affected traffic. Note that our problem statement does not directly target root cause analysis, nor automated remedies to the detected failures.

FANcY. We present FANcY, a gray failure detector tailored to ISPs. FANcY relies on an inter-switch protocol enabling data planes to synchronize packet counters and detect packet losses by comparing the values of those counters. Counters provide the minimal information needed to localize gray failures; frequently exchanging them provides detection speed and scalability (e.g., consumed memory).

We used P4 to…

Enable programmable packet scheduling at Tbps

USENIX NSDI 2020

SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues

Albert Gran Alcoz
ETH Zürich

Alexander Dietmüller
ETH Zürich

Laurent Vanbever
ETH Zürich

Abstract

Push-In First-Out (PIFO) queues are hardware primitives which enable programmable packet scheduling by providing the abstraction of a priority queue at line rate. However, implementing them at scale is not easy: just hardware designs (not implementations) exist, which support only about 1k flows.

In this paper, we introduce SP-PIFO, a programmable packet scheduler which closely approximates the behavior of PIFO queues using strict-priority queues—*at line rate, at scale, and on existing devices*. The key insight behind SP-PIFO is to dynamically adapt the mapping between packet ranks and available strict-priority queues to minimize the scheduling errors with respect to an ideal PIFO. We present a mathematical formulation of the problem and derive an adaptation technique which closely approximates the optimal queue mapping without any traffic knowledge.

We fully implement SP-PIFO in P4 and evaluate it on real workloads. We show that SP-PIFO: (i) closely matches PIFO, with as little as 8 priority queues; (ii) scales to large amount of flows and ranks; and (iii) quickly adapts to traffic variations. We also show that SP-PIFO runs at line rate on existing hardware (Barefoot Tofino), with a negligible memory footprint.

1 Introduction

Until recently, packet scheduling was one of the last bastions standing in the way of complete data-plane programmability. Indeed, unlike forwarding whose behavior can be adapted thanks to languages such as P4 [7] and reprogrammable hardware [2], scheduling behavior is mostly set in stone with hardware implementations that can, at best, be configured.

To enable programmable packet scheduling, the main challenge was to find an appropriate abstraction which is flexible enough to express a wide variety of scheduling algorithms and yet can be implemented efficiently in hardware [22]. In [23], Sivaraman et al. proposed to use Push-In First-Out (PIFO) queues as such an abstraction. PIFO queues allow enqueued packets to be pushed in arbitrary positions (according to the packets rank) while being drained from the head.

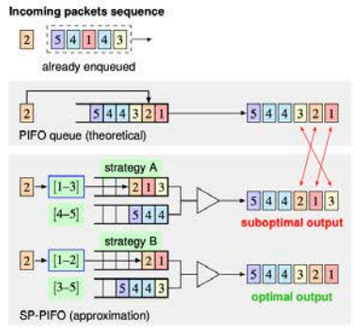


Figure 1: SP-PIFO approximates the behavior of PIFO queues by adapting how packet ranks are mapped to priority queues.

While PIFO queues enable programmable scheduling, implementing them in hardware is hard due to the need to arbitrarily sort packets at line rate. [23] described a possible hardware design (not implementation) supporting PIFO on top of Broadcom Trident II [1]. While promising, realizing this design in an ASIC is likely to take years [6], not including deployment. Even ignoring deployment considerations, the design of [23] is limited as it only supports ~1000 flows and relies on the assumption that the packet ranks increase monotonically within each flow, which is not always the case.

Our work In this paper, we ask whether it is possible to approximate PIFO queues at scale, in existing programmable data planes. We answer positively and present SP-PIFO, an adaptive scheduling algorithm that closely approximates PIFO behaviors on top of widely-available Strict-Priority (SP) queues. The key insight behind SP-PIFO is to dynamically adapt the mapping between packet ranks and SP queues in order to minimize the amount of scheduling mistakes relative to a hypothetical ideal PIFO implementation.

We used P4 to…

Converge upon
remote Internet failures
in much less than 1 sec

USENIX NSDI 2019

Blink: Fast Connectivity Recovery Entirely in the Data Plane

Thomas Holterbach,* Edgar Costa Molero,* Maria Apostolaki*
Alberto Dainotti,† Stefano Vissicchio,‡ Laurent Vanbever*

*ETH Zurich, †CAIDA / UC San Diego, ‡University College London

Abstract

We present Blink, a data-driven system that leverages TCP-induced signals to detect failures directly in the data plane. The key intuition behind Blink is that a TCP flow exhibits a predictable behavior upon disruption: retransmitting the same packet over and over, at epochs exponentially spaced in time. When compounded over multiple flows, this behavior creates a strong and characteristic failure signal. Blink efficiently analyzes TCP flows to: *(i)* select which ones to track; *(ii)* reliably and quickly detect major traffic disruptions; and *(iii)* recover connectivity—all this, completely in the data plane.

We present an implementation of Blink in P4 together with an extensive evaluation on real and synthetic traffic traces. Our results indicate that Blink: *(i)* achieves sub-second rerouting for large fractions of Internet traffic; and *(ii)* prevents unnecessary traffic shifts even in the presence of noise. We further show the feasibility of Blink by running it on an actual Tofino switch.

1 Introduction

Thanks to widely deployed fast-convergence frameworks such as IPFFR [35], Loop-Free Alternate [7] or MPLS Fast Reroute [29], sub-second and ISP-wide convergence upon link or node failure is now the norm [6, 15]. At a high-level, these fast-convergence frameworks share two common ingredients: (i) *fast detection* by leveraging hardware-generated signals (*e.g.,* Loss-of-Light or unanswered hardware keepalive [23]); and (ii) *quick activation* by promptly activating pre-computed backup state upon failure instead of recomputing the paths on-the-fly.

Problem: Convergence upon *remote* failures is still slow. These frameworks help ISPs to retrieve connectivity upon *internal* (or peering) failures but are of no use when it comes to restoring connectivity upon *remote* failures. Unfortunately, remote failures are both frequent and slow to repair, with average convergence times above 30 s [19, 24, 28]. These failures indeed trigger a *control-plane-driven* convergence through the propagation of BGP updates on a per-router and per-prefix basis. To reduce convergence time, SWIFT [19] predicts the entire extent of a remote failure from a few received BGP updates, leveraging the fact that such updates are correlated (*e.g.,* they share the same AS-PATH). The fundamental problem with SWIFT though, is that it can take $O(minutes)$ for the *first* BGP update to propagate after the corresponding data-plane failure.

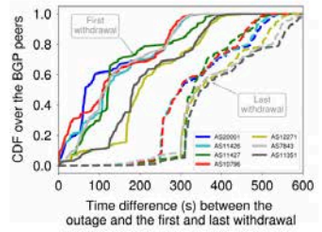


Figure 1: It can take minutes to receive the *first* BGP update following data-plane failures during which traffic is lost.

We illustrate this problem through a case study, by measuring the time the *first* BGP updates took to propagate after the Time Warner Cable (TWC) networks were affected by an outage on August 27 2014 [1]. We consider as outage time t_0, the time at which traffic originated by TWC ASes observed at a large darknet [10] suddenly dropped to zero. We then collect, for each of the routers peering with RouteViews [27] and RIPE RIS [2], the timestamp t_1 of the first BGP withdrawal they received from the same TWC ASes. Figure 1 depicts the CDFs of $(t_1 - t_0)$ over all the BGP peers (100+ routers, in most cases) that received withdrawals for 7 TWC ASes: more than half of the peers took *more than a minute* to receive the first update (continuous lines). In addition, the CDFs of the time difference between the outage and the *last* prefix withdrawal for each AS, show that BGP convergence can be as slow as several minutes (dashed lines).

We used P4 to…

Protect Bitcoin against DDoS attacks

in hardware

NDSS 2019

SABRE: Protecting Bitcoin against Routing Attacks

Maria Apostolaki
ETH Zurich
apmaria@ethz.ch

Gian Marti
ETH Zurich
gimarti@student.ethz.ch

Jan Müller
ETH Zurich
jan.m.muller@me.com

Laurent Vanbever
ETH Zurich
lvanbever@ethz.ch

***Abstract*—Nowadays Internet routing attacks remain practically effective as existing countermeasures either fail to provide protection guarantees or are not easily deployable. Blockchain systems are particularly vulnerable to such attacks as they rely on Internet-wide communications to reach consensus. In particular, Bitcoin—the most widely-used cryptocurrency—can be split in half by any AS-level adversary using BGP hijacking.**

In this paper, we present SABRE, a secure and scalable Bitcoin relay network which relays blocks worldwide through a set of connections that are resilient to routing attacks. SABRE runs alongside the existing peer-to-peer network and is easily deployable. As a critical system, SABRE design is highly resilient and can efficiently handle high bandwidth loads, including Denial of Service attacks.

We built SABRE around two key technical insights. First, we leverage fundamental properties of inter-domain routing (BGP) policies to host relay nodes: *(i)* in networks that are inherently protected against routing attacks; and *(ii)* on paths that are economically-preferred by the majority of Bitcoin clients. These properties are generic and can be used to protect other Blockchain-based systems. Second, we leverage the fact that relaying blocks is communication-heavy, not computation-heavy. This enables us to offload most of the relay operations to programmable network hardware (using the P4 programming language). Thanks to this hardware/software co-design, SABRE nodes operate seamlessly under high load while mitigating the effects of malicious clients.

We present a complete implementation of SABRE together with an extensive evaluation. Our results demonstrate that SABRE is effective at securing Bitcoin against routing attacks, even with deployments of as few as 6 nodes.

I. Introduction

Cryptocurrencies, and Bitcoin in particular, are vulnerable to routing attacks in which network-level attackers (i.e., malicious Autonomous Systems or ASes) manipulate routing (BGP) advertisements to divert their connections. Once on-path, the AS-level attacker can disrupt the consensus algorithm by partitioning the peer-to-peer network. Recent studies [17] have shown that these attacks are practical and disruptive. Specifically, *any* AS-level attacker can isolate ~50% of the Bitcoin mining power by hijacking less than 100 prefixes [17]. Such attacks can lead to significant revenue loss for miners and enable exploits such as double spending.

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23252
www.ndss-symposium.org

Problem Protecting against such partitioning attacks is challenging. On the one hand, local (and easily deployable) countermeasures [17] fail to provide strong protection guarantees. These countermeasures include having Bitcoin clients monitor their connections (e.g., for increased or abnormal delays) or having them select their peers based on routing information. On the other hand, Internet-wide countermeasures are extremely hard to deploy. For example, systematically hosting Bitcoin clients in /24 prefixes (to prevent more-specific prefix attacks) requires the unlikely cooperation of all Internet Service Providers hosting Bitcoin clients in addition to a considerable increase to the size of the Internet routing tables. Worse yet, even heavy protocol modification such as encrypting *all* Bitcoin traffic would not be enough to guarantee Bitcoin safety as AS-level attackers would still be able to distinguish (and drop) Bitcoin traffic using transport headers.

SABRE: A Secure Relay Network for Bitcoin In this paper, we present SABRE, a secure relay network which runs alongside the existing Bitcoin network and which can protect the vast majority of the Bitcoin clients against routing attacks. Unlike existing countermeasures, SABRE secures Bitcoin against routing attacks in a way which: *(i)* provides strong security guarantees to any connected client by enabling it to learn and propagate blocks; *(ii)* is partially deployable; and *(iii)* provides security benefits early-on in the deployment, with as little as two relay nodes. We built SABRE based on two key insights.

Insight #1: Hosting relays in inherently safe locations Our first insight is to host SABRE relay nodes in locations that: *(i)* prevent attackers from diverting relay-to-relay connections, so as to secure SABRE internal connectivity; and *(ii)* are attractive (from a routing viewpoint) to many Bitcoin clients, so as to protect client connections to the relay network. We do so by leveraging a fundamental characteristic of BGP policies, namely, that connections established between two ASes which directly peer with each other and which have no customers cannot be diverted by routing attacks. In SABRE, only such ASes are considered for relay locations.

Using real routing data, we show that such safe locations are plentiful in the current Internet with 2000 ASes being eligible. These ASes include large cloud providers, content delivery networks, and Internet eXchange Points which already provide hosting services today and therefore have an incentive to host SABRE nodes. We also show that SABRE deployments with 6 nodes are already enough to protect 80% of the clients from 96% of the AS-level adversaries (assuming worst case scenario for SABRE).

We used P4 to…

Speed-up
network computations
offloading them to hardware

Hotnets 2018

Hardware-Accelerated Network Control Planes

Edgar Costa Molero
ETH Zürich
cedgar@ethz.ch

Stefano Vissicchio
University College London
s.vissicchio@cs.ucl.ac.uk

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

ABSTRACT

One design principle of modern network architecture seems to be set in stone: a software-based control plane drives a hardware- or software-based data plane. We argue that it is time to revisit this principle after the advent of programmable switch ASICs which can run complex logic at line rate.

We explore the possibility and benefits of accelerating the control plane by offloading some of its tasks directly to the network hardware. We show that programmable data planes are indeed powerful enough to run key control plane tasks including: failure detection and notification, connectivity retrieval, and even policy-based routing protocols. We implement in P4 a prototype of such a "hardware-accelerated" control plane, and illustrate its benefits in a case study.

Despite such benefits, we acknowledge that offloading tasks to hardware is not a silver bullet. We discuss its tradeoffs and limitations, and outline future research directions towards hardware-software codesign of network control planes.

1 INTRODUCTION

As the "brain" of the network, the control plane is one of its most important assets. Among other things, the control plane is responsible for *sensing* the status of the network (e.g., which links are up or which links are overloaded), *computing* the best paths along which to guide traffic, and *updating* the underlying data plane accordingly. To do so, the control plane is composed of many dynamic and interacting processes (e.g., routing, management and accounting protocols) whose operation must scale to large networks. In contrast, the data plane is "only" responsible for forwarding traffic according to the control plane decisions, albeit as fast as possible.

These fundamental differences lead to very different design philosophies. Given the relative simplicity of the data plane and the "need for speed", it is typically entirely implemented in hardware. That said, software-based implementations of data planes are also commonly found (e.g., OpenVSwitch [30]) together with hybrid software-hardware ones (e.g., CacheFlow [20]). In short, data plane implementations cover the entire implementation spectrum, from pure software to pure hardware. In contrast, there is *much* less diversity in control plane implementations. The sheer complexity of the control plane tasks (e.g., performing routing computations) together with the need to update them relatively frequently (e.g., to support new protocols and features) indeed calls for software-based implementations, with only a few key tasks (e.g., detecting physical failures, activating backup forwarding state) being (sometimes) offloaded to hardware [13, 22].

Yet, we argue that a number of recent developments are creating both the *need* and *opportunity* for rethinking basic design and implementation choices of network control planes.

Need There is a growing need for faster, more scalable, and yet more powerful control planes. Nowadays, even beefed-up and highly-optimized software control planes can only process thousands of (BGP) control plane messages per second [23], and can take *minutes* to converge upon large failures [17, 36]. Parallelizing only marginally helps: for instance, the BGP specification [31] mandates to lock all Adj-RIBs-In before proceeding with the best-path calculation, essentially preventing the parallel execution of best path computations. A concrete risk is that convergence time will keep increasing with the network size and the number of Internet destinations. At the same time, recent research has repeatedly shown the performance benefits of controlling networks with extremely tight control loops, among others to handle congestion (e.g., [7, 21, 29]).

Opportunity Modern reprogrammable switches (e.g., [1]) can perform complex stateful computations on billions of packets per second [19]. Running (pieces of) the control plane at such speeds would lead to almost "instantaneous" convergence, leaving the propagation time of the messages as the primary bottleneck. Besides speed, offloading control plane tasks to hardware would also help by making them traffic-aware. For instance, it enables to update forwarding entries consistently with real-time traffic volumes rather than in a random order.

Research questions Given the opportunity and the need, we argue that it is time to revisit the control plane's design and implementation by considering the problem of offloading parts of it to hardware. This redesign opens the door to multiple research questions including: *Which pieces of the control plane should be offloaded? What are the benefits?* and *How can we overcome the fundamental hardware limitations?* These fundamental limitations come mainly from the very limited instruction set (e.g., no floating point) and the memory available (i.e., around tens of megabytes [19]) of programmable network hardware. We start to answer these questions in this paper and make two contributions.

HotNets-XVII, November 15–16, 2018, Redmond, WA, USA

ACM ISBN 978-1-4503-6120-0/18/11…$15.00
https://doi.org/10.1145/3286062.3286080

We used P4 to…

Obfuscate network traffic from curious onlookers

NDSS 2022

ditto: WAN Traffic Obfuscation at Line Rate

Roland Meier
ETH Zürich
meierrol@ethz.ch

Vincent Lenders
armasuisse
vincent.lenders@ar.admin.ch

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

***Abstract*—Many large organizations operate dedicated wide area networks (WANs) distinct from the Internet to connect their data centers and remote sites through high-throughput links. While encryption generally protects these WANs well against content eavesdropping, they remain vulnerable to traffic analysis attacks that infer visited websites, watched videos or contents of VoIP calls from analysis of the traffic volume, packet sizes or timing information. Existing techniques to obfuscate Internet traffic are not well suited for WANs as they are either highly inefficient or require modifications to the communication protocols used by end hosts.**

This paper presents ditto, a traffic obfuscation system adapted to the requirements of WANs: achieving high-throughput traffic obfuscation at line rate without modifications of end hosts. ditto adds padding to packets and introduces chaff packets to make the resulting obfuscated traffic independent of production traffic with respect to packet sizes, timing and traffic volume.

We evaluate a full implementation of ditto running on programmable switches in the network data plane. Our results show that ditto runs at 100 Gbps line rate and performs with negligible performance overhead up to a realistic traffic load of 70 Gbps per WAN link.

I. Introduction

Many large organizations operate dedicated wide area networks (WANs) as a critical infrastructure distinct from the Internet to connect their data centers and remote sites. For example, cloud service providers such as Google [66], Amazon [23], and Microsoft [30] operate WANs to achieve low-latency, high-throughput inter data center communication. Public safety and security organizations rely on WANs to achieve secure and reliable communication between their sites (e.g., [7], [17], [20], [28], [65]). For large organizations, these WANs provide 100s of Gbps to Tbps of capacity over long distances and can cost 100s of millions of dollars per year [63].

WANs are an attractive target for eavesdropping attacks and mass surveillance because they are often used to transport large amounts of sensitive data. And because WANs spread over large geographical areas, it is impossible to secure the cables physically from wiretapping. Past revelations show that intercontinental fiber links were subject to tapping by governmental agencies [27], [59] or other entities [77] and many devices are available to tap on fiber links [11], [25], [47], [68], [85]. Indeed, major operators such as Amazon, Microsoft, and OVH acknowledge that WAN traffic is at risk and they use MACsec [21] to encrypt their traffic not only at the application layer, but also at the link layer [29], [35], [76], [82].

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.24056
www.ndss-symposium.org

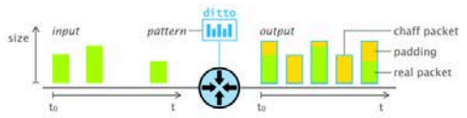


Fig. 1: ditto adds padding and chaff packets such that the outgoing traffic always follows a predefined pattern

However, it is well known that encryption alone is not sufficient to protect against traffic analysis attacks [53], [64]. Even if the network traffic is end-to-end encrypted, metadata such as the traffic volume, the packet sizes and the timing information reveals a lot about ongoing activities. As a result, eavesdroppers intercepting the WAN communication can still perform traffic analysis attacks. Such attacks are mostly known from Internet traffic, where past work shows that it is possible to infer the contents of VoIP calls [31], [46], streamed movies [50], [88]; visited websites [37], [61], [81], [91], [101], or the device identities [22], [24], [78], [83], [87], [93] without having to break the encryption. However, the same attacks can be applied to WAN traffic if the WAN carries the incoming and outgoing Internet traffic (which is typically the case if a company sends all Internet traffic via a central firewall). More generally, it has been shown many times (e.g., in [44], [49], [89]) that traffic classification also works for encrypted traffic.

Many techniques have been proposed to protect against traffic analysis attacks in the Internet. However, these techniques are not well adapted to the specific requirements of WAN traffic protection. Techniques such as BuFLO [22], CS-BuFLO [36], HORNET [42], or TARANET [43] add padding to obfuscate the size of individual packets and flows and require modifications on the software and protocols of the end hosts. For many organizations operating a WAN, it is impossible to adapt these protocols on all end hosts (e.g., because a cloud provider does not control the software that is running on its customer's instances). Other techniques such as Loopix [84], PriFi [32], or Wang et al. [98] impose strict transmission schedules and rates per flow, and thus severely limit the achievable throughput. As WAN traffic is high-throughput in nature, these solutions are not efficient enough to deal with high link traffic rates up to 100 Gbps.

This paper presents **ditto**, an in-network and hardware-based traffic obfuscation system specifically tailored to WANs. As illustrated in Fig. 1, ditto shapes traffic according to a predefined pattern (a periodic sequence of packet sizes at a fixed rate) using three operations: *(i)* packet padding; *(ii)* packet delaying; and *(iii)* chaff packet insertion. When there are "real" packets to transmit, ditto pads and transmits them. When there are no real packets, it transmits dummy "chaff" packets. Therefore, ditto only adds overhead (padding

We used P4 to…

Obfuscate network topologies from attackers

Usenix Security 2018

NetHide: Secure and Practical Network Topology Obfuscation

Roland Meier*, Petar Tsankov*, Vincent Lenders◇, Laurent Vanbever*, Martin Vechev*

* *ETH Zürich* ◇*armasuisse*

nethide.ethz.ch

Abstract

Simple path tracing tools such as `traceroute` allow malicious users to infer network topologies remotely and use that knowledge to craft advanced denial-of-service (DoS) attacks such as Link-Flooding Attacks (LFAs). Yet, despite the risk, most network operators still allow path tracing as it is an essential network debugging tool.

In this paper, we present NetHide, a network topology obfuscation framework that mitigates LFAs while preserving the practicality of path tracing tools. The key idea behind NetHide is to formulate network obfuscation as a multi-objective optimization problem that allows for a flexible tradeoff between security (encoded as hard constraints) and usability (encoded as soft constraints). While solving this problem exactly is hard, we show that NetHide can obfuscate topologies at scale by only considering a subset of the candidate solutions and without reducing obfuscation quality. In practice, NetHide obfuscates the topology by intercepting and modifying path tracing probes directly in the data plane. We show that this process can be done at line-rate, in a stateless fashion, by leveraging the latest generation of programmable network devices.

We fully implemented NetHide and evaluated it on realistic topologies. Our results show that NetHide is able to obfuscate large topologies (> 150 nodes) while preserving near-perfect debugging capabilities. In particular, we show that operators can still precisely trace back $> 90\,\%$ of link failures despite obfuscation.

1 Introduction

Botnet-driven Distributed Denial-of-Service (DDoS) attacks constitute one of today's major Internet threats [1, 2, 5, 10]. Such attacks can be divided in two categories depending on whether they target end-hosts and services (volume-based attacks) or the network infrastructure itself (link-flooding attacks, LFAs).

Volume-based attacks are the simplest and work by sending massive amounts of data to selected targets. Recent examples include the 1.2 Tbps DDoS attack against Dyn's DNS service [6] in October 2016 and the 1.35 Tbps DDoS attack against GitHub in February 2018 [8]. While impressive, these attacks can be mitigated today by diverting the incoming traffic through large CDN infrastructures [23]. As an illustration, CloudFlare's infrastructure can now mitigate volume-based attacks reaching Terabits per second [18].

Link-flooding attacks (LFAs) [26, 38] are more sophisticated and work by having a botnet generate low-rate flows between pairs of bots or towards public services such that all of these flows cross a given set of network links or nodes, degrading (or even preventing) the connectivity for *all* services using them. LFAs are much harder to detect as: *(i)* traffic volumes are relatively small (10 Gbps or 40 Gbps attacks are enough to kill most Internet links [31]); and *(ii)* attack flows are indistinguishable from legitimate traffic. Representative examples include the Spamhaus attack which flooded selected Internet eXchange Point (IXP) links in Europe and Asia [4, 7, 12].

Unlike volume-based attacks, performing an LFA requires the attacker to know the topology *and* the forwarding behavior of the targeted network. Without this knowledge, an attacker can only "guess" which flows share a common link, considerably reducing the attack's efficiency. As an illustration, our simulations indicate that congesting an *arbitrary* link without knowing the topology requires 5 times more flows, while congesting a *specific* link is order of magnitudes more difficult.

Nowadays, attackers can easily acquire topology knowledge by running path tracing tools such as `traceroute` [17]. In fact, previous studies have shown that entire topologies can be precisely mapped with `traceroute` provided enough vantage points are used [37], a requirement easily met by using large-scale measurement platforms (e.g., RIPE Atlas [16]).

Advanced Topics in Communication Networks

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich

Tue 20 Sep 2022