Communication Networks

Prof. Laurent Vanbever

Communication Networks

Spring 2019

Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

May 13 2019

Materials inspired from Olivier Bonaventure (University of Louvain)

Last week on

Communication Networks

E-mail	2nd project
MX, SMTP, POP, IMAP	Introduction

We'll study e-mail from three different perspectives

Content	Infrastructure/ Transmission	Retrieval
Format: Header/Content **Encoding:** MIME	**SMTP:** Simple Mail Transfer Protocol **Infrastructure** mail servers	**POP:** Post Office Protocol **IMAP:** Internet Message Access Protocol

Content	Infrastructure/ Transmission	Retrieval
Format: Header/Content **Encoding:** MIME		

A **header**, in 7-bit U.S. ASCII text

Header

From: Laurent Vanbever <lvanbever@ethz.ch>
To: Tobias Buehler <buehlert@ethz.ch>
Subject: [comm-net] Exam questions

A **body**, also in 7-bit U.S. ASCII text

From: Laurent Vanbever <lvanbever@ethz.ch>
To: Tobias Buehler <buehlert@ethz.ch>
Subject: [comm-net] Exam questions

Body

Hi Tobias,

Here are some interesting questions…

Best,
Laurent

Email relies on 7-bit U.S. ASCII...
How do you send non-English text? Binary files?

Solution — **Multipurpose Internet Mail Extensions**

commonly known as MIME, standardized in RFC 822

Content	Infrastructure/ Transmission	Retrieval

SMTP: Simple Mail Transfer Protocol

Infrastructure
mail servers

An e-mail address is composed of two parts identifying the local mailbox and the domain

lvanbever @ ethz.ch

lvanbever → local mailbox

ethz.ch → domain name

actual **mail server** is identified using a DNS query asking for **MX records**

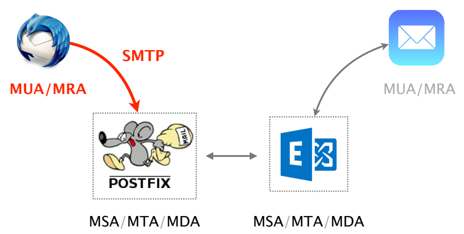
Simple Mail Transfer Protocol (SMTP) is the current standard for transmitting e-mails

SMTP is a text-based, client-server protocol
client sends the e-mail, server receives it

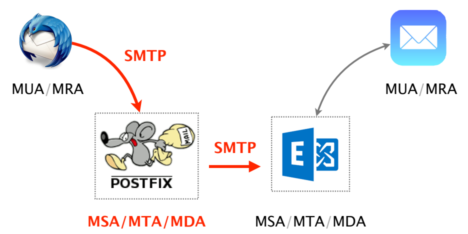
SMTP uses reliable data transfer
built on top of TCP (port 25 and 465 for SSL/TLS)

SMTP is a push-like protocol
sender pushes the file to the receiving server (no pull)

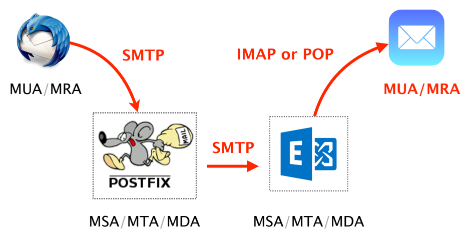
The sender MUA uses SMTP to transmit the e-mail first to a local MTA (e.g. mail.ethz.ch, gmail.com, hotmail.com)



The local MTA then looks up the MTA of the recipient domain (DNS MX) and transmits the e-mail further



Once the e-mail is stored at the recipient domain, IMAP or POP is used to retrieve it by the recipient MUA



Content	Infrastructure/ Transmission	Retrieval

POP: Post Office Protocol
IMAP: Internet Message Access Protocol

This week on Communication Networks

IPv6

next generation of Internet addressing

programmable networks

next generation of network devices

IPv6

next generation of Internet addressing

programmable networks

The long way from...

World population: 7.5 billion

~0.6 IPv4 addresses per person

...to...

Average # of atoms in a human: 6.10^{27}

~7.5 IPv6 addresses per "human" atom

First, let's look at some history

late 1980s	Exponential growth of the Internet
1992	Most class B networks have been assigned experts warn that IPv4 addresses might run out
1993	Introduction of classless IPv4 addresses
1994	"Address Allocation for Private Internets" 3 reserved IPv4 blocks for private networks Hosts in private IP space are unreachable from Internet

---

IPv6 originally appeared in 1998

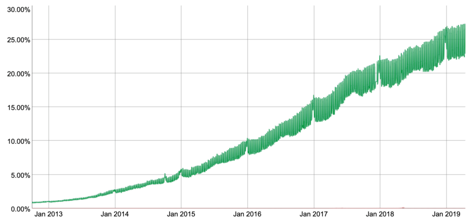
i.e. more than 20 years ago

1994 (cont'd)	"IP Network Address Translator (NAT)" A public address is mapped to an entire private IP space
1998	IETF standardization of the IPv6 draft
2005	Estimated timeframe for massive adaption of IPv6 Did not happen...
2008	It is possible to resolve domain names using IPv6 only

... and is now *finally* picking up steam

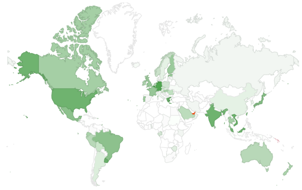
2011	Last unassigned top-level IPv4 block is distributed All major operating systems have stable IPv6 support Support for mobile devices varies
2012	World IPv6 Launch day A large number of content and ISPs permanently enable IPv6
2018	>20% of Google traffic is on IPv6 with wide differences across countries

Almost of third of the requests seen by Google are done using IPv6



https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption

Not all countries are equivalent though



The darker the green,
the larger the deployment

top country: Belgium
(53% deployment)

https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption

Thus far IPv4 has been very persistent, and that's quite understandable

Deploying IPv6 require every device to support it
All routers, middleboxes, end hosts, applications, …

Most of IPv6 new features were back-ported to IPv4
No obvious advantage in using IPv6

Network Address Translation is working well
The pain of address depletion is not obvious

Network Address Translation (NAT)

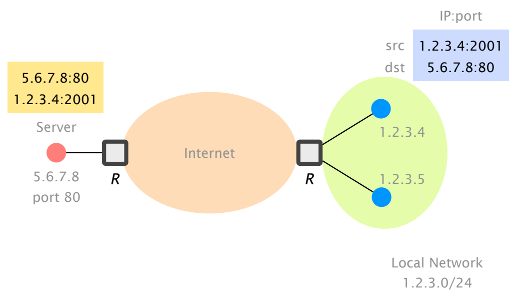
Sharing a single (public) address between hosts
Port numbers (transport layer) are used to distinguish

One of the main reasons why we can still use IPv4
Saved us from address depletion

Violates the general end-to-end principle of the Internet
A NAT box adds a layer of indirection

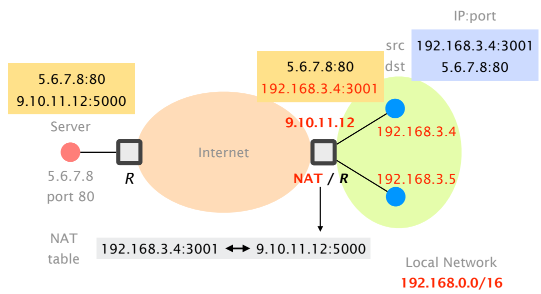
The Internet before NAT

Every machine connected to the Internet had a unique IP



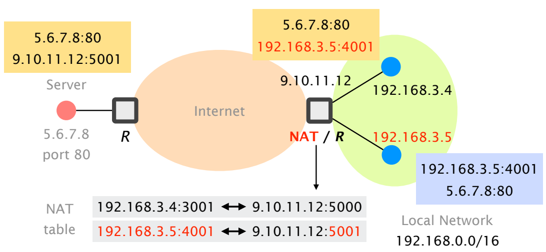
The Internet with NAT

Hosts behind NAT get a private address



The Internet with NAT

The port numbers are used to multiplex single addresses



NAT also provides other (dis-)advantages

Better privacy/anonymization
All hosts in one network get the same public IP
But, cookies, browser version, … still identify hosts

Better security
From the outside you cannot directly reach the hosts
Problematic e.g., for online gaming

Limited scalability (size of the mapping table)
Example: Wi-Fi access problems in public places
(e.g., lecture hall) often due to a full NAT table

Let's talk about **IPv6**

IPv6 addresses are encoded in 128 bits

Notation	8 groups of 16 bits each separated by colons (:) Each group is written as four hexadecimal digits
Simplification	Leading zeros in any group are removed **One** section of zeros is replaced by a double colon (::) Normally the longest section
Examples	1080:0:0:0:8:800:200C:417A → 1080::8:800:200C:417A FF01:0:0:0:0:0:0:0101 → FF01::101 0:0:0:0:0:0:0:1 → ::1

There are three types of IPv6 addresses: unicast, anycast, and multicast

Unicast	Identifies a single interface Packets are delivered to this specific interface
Anycast	Identifies a set of interfaces Packets are delivered to the "nearest" interface
Multicast	Identifies a set of interfaces Packets are delivered to **all** interfaces

Unicast	Identifies a single interface Packets are delivered to this specific interface

Global unicast addresses are hierarchically allocated

similar to global IPv4 addresses

128 bits

N bits	M bits	128-N-M bits
global routing prefix	subnet ID	Interface ID
Identifies the ISP responsible for the address	A subnet in this ISP or a customer of the ISP	Usually 64 bits Based on the MAC address

Allocation of IPv6 (global unicast) addresses

iana
Internet Assigned Numbers Authority

The Internet Assigned Numbers Authority (IANA) assigns blocks to Regional IP address Registries (RIR)
For example RIPE, ARIN, APNIC, …

Currently, only 2000::/3 is used for global unicast
All addresses are in the range of 2000 to 3FFF

Link-local addresses are unique to a single link (subnet)

same as private IPv4 addresses

128 bits

10 bits	54 bits	64 bits
FE80	0000…0000	Interface ID

Each host/router **must** generate a link-local address for **each** of its interfaces
An interface therefore can have multiple IPv6 addresses

In addition to global and link-local addresses, some IPv6 unicast addresses have a special meaning

Unspecified address	0:0:0:0:0:0:0:0 Used as src address if no IPv6 address available
Loopback address	0:0:0:0:0:0:0:1 → ::1 127.0.0.1 for IPv4 addresses
IPv4 embedded	The lowest 32 bits contains an IPv4 address useful when deploying IPv6
Important	There are no IPv6 broadcast addresses

Anycast Identifies a set of interfaces

Packets are delivered to the „nearest" interface

IPv6 anycast addresses

Multiple interfaces with the same address

Packets are sent to the nearest interface

Anycast use the global unicast address range

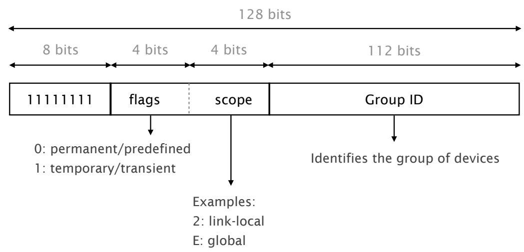
E.g. for DNS or HTTP services

IPv6 anycast is rarely used

Multicast Identifies a set of interfaces

Packets are delivered to **all** interfaces

Multicast addresses identify a group of receivers/interfaces



Some multicast addresses are well-known and used for auto-discovery, bootstraping, etc.

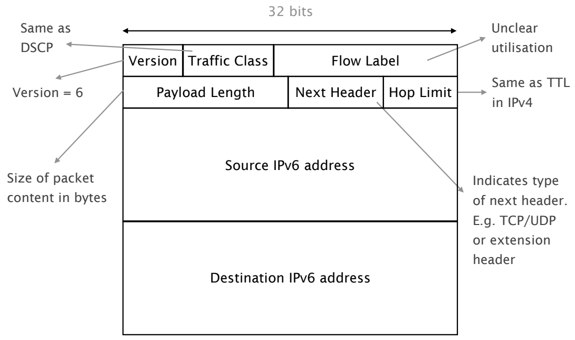
FF02::1 — All IPv6 end-systems

E.g. hosts, servers, routers, mobile devices, …

FF02::2 — All IPv6 routers

All routers automatically belong to this group

The IPv6 packet header format



Compared to IPv4, IPv6 does…

not **include checksums in the packet header**

link, transport or application layer provide checksums

not **support fragmentation**

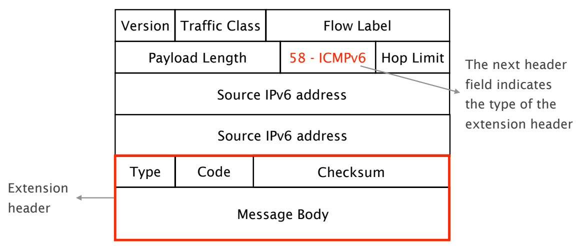
End host is required to send small enough packets

provide more flexibility

flow labels and extension headers

Extension header example: ICMPv6

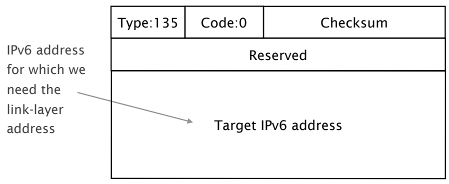
Similar functions than IPv4 ICMP



ICMPv6 can be used for neighbor discovery

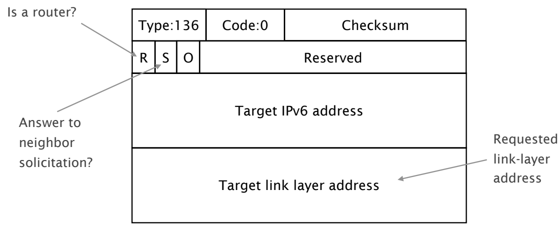
replacement for IPv4's ARP

First step: neighbor solicitation



ICMPv6 can be used for neighbor discovery

Second step: neighbor advertisement



How can a node obtain its IPv6 address(es)?

Manual configuration
As in the project, e.g. with ifconfig

From a server by using DHCPv6
Similar to the IPv4 version

Automatically
Using its link-local address and neighbor discovery

IPv6 autoconfiguration to find link-local address

Consider an end-system which has just started, it needs an IPv6 address to send ICMPv6 messages

Ethernet (MAC): 0800:200C:417A
Link-local: FE80::M_{64}(800:200C:417A)
M_{64}: 64-bit representation of the MAC address

Neighbor solicitation for FE80::M_{64}(800:200C:417A)
If **no** answer, the created link-local address is valid

IPv6 autoconfiguration to obtain the IPv6 prefix of subnet

Routers periodically advertise the prefix
Sent to all end-systems: FF02::1

The advertisements can contain:
IPv6 prefix and length
Network MTU to use
Maximum hop limit to use
Lifetime of the default router
How long generated addresses are preferred

IPv6 autoconfiguration to build global unicast address

Ethernet (MAC): 0800:200C:417A

Prefix: 2001:6a8:3080:1::/64

Global unicast:
2001:6a8:3080:1:M_{64}(800:200C:417A)

contains MAC address of host

To port your IPv4-based application to IPv6, you need to…

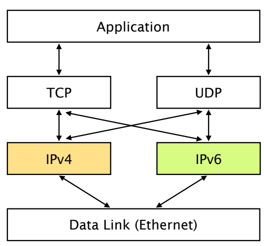
change the used socket functions

adjust all logging functions

adapt all data structures to support IPv6 addresses

adjust user interface elements to display IPv6

Today, a lot of applications and OSes use a dual stack approach



Over the years, a lot of transition mechanisms were developed

- 6in4
- 6to4
- Teredo
- SIIT
- 6rd
- GRE
- AYiYA
- …

Tunnel IPv6 packets over static IPv4 links (6in4)

IPv4 payload	Application
	TCP header and ports — cannot cross NATs: TCP/UDP header not available
	Version IPv6 Src/dst: IPv6 addresses Protocol: 6 (TCP)
IPv4 header	Version: IPv4 Src/dst IPs: IPv4 addresses Protocol: 41 (IPv6)

IPv6 @ home (Swisscom Internet access box)

You will be assigned an IPv4 and IPv6 address

IPv6

programmable networks

next generation of network devices

Networking is on the verge of a paradigm shift towards *deep* programmability

Network programmability is attracting tremendous industry interest (and money)

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

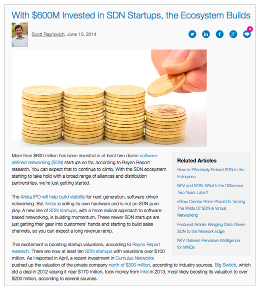
JULY 23, 2012 AT 1:25 PM PT

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.

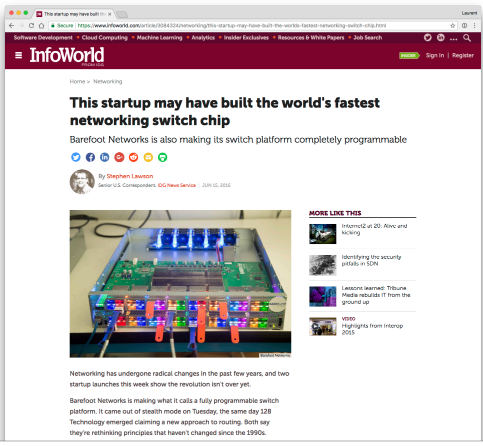
The news comes on the same day that VMware was to report quarterly earnings. And while I don't usually cover VMware's earnings, I may as well mention the results: The company reported revenue for the quarter ended June rose to $1.12 billion, while earnings on a per-share basis were 68 cents. Analysts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



InfoWorld

Home > Networking

This startup may have built the world's fastest networking switch chip

Barefoot Networks is also making its switch platform completely programmable

By Stephen Lawson
Senior U.S. Correspondent, IDG News Service | JUN 15, 2016

MORE LIKE THIS
- Internet2 at 20: Alive and kicking
- Identifying the security pitfalls in SDN
- Lessons learned: Tribune Media rebuilds IT from the ground up
- VIDEO Highlights from Interop 2015

Networking has undergone radical changes in the past few years, and two startup launches this week show the revolution isn't over yet.

Barefoot Networks is making what it calls a fully programmable switch platform. It came out of stealth mode on Tuesday, the same day 128 Technology emerged claiming a new approach to routing. Both say they're rethinking principles that haven't changed since the 1990s.

Network programmability is getting traction in many academic communities

Networking	Systems	Distributed Algorithms	Security	PL
SIGCOMM	OSDI	PODC	CCS	PLDI
NSDI	SOSP	DISC	NDSS	POPL
HotNets	SOCC		Usenix Security	OOPSLA
CoNEXT			S&P	

>7.7k

# of citations of the original OpenFlow paper (*) in ~10 years

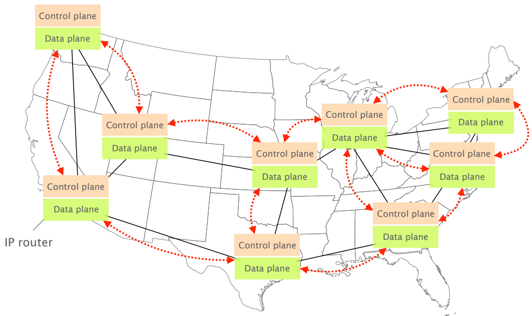
(*) https://dl.acm.org/citation.cfm?id=1355746

Why? It's really a story in 3 stages

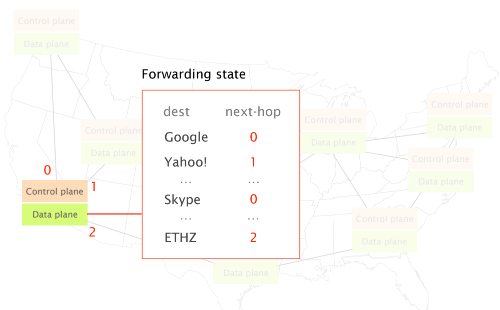
Stage 1

The network management crisis

Networks are large distributed systems running a set of distributed algorithms



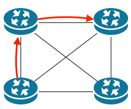
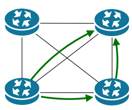
These algorithms produce the forwarding state which drives IP traffic to its destination



dest	next-hop
Google	0
Yahoo!	1
...	...
Skype	0
...	...
ETHZ	2

Operators adapt their network forwarding behavior by configuring each network device individually

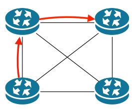
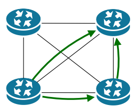
---

Given  and 

an existing network behavior induced by a low-level configuration C

a desired network behavior

Adapt C so that the network follows the new behavior

Given  and 

an existing network behavior induced by a low-level configuration C

a desired network behavior

Adapt C so that the network follows the new behavior

Configuring each element is often done manually,
using arcane low-level, vendor-specific "languages"

Cisco IOS

```
!
ip multicast-routing
!
interface Loopback0
 ip address 120.1.7.7 255.255.255.255
 ip ospf 1 area 0
!
!
interface Ethernet0/0
 no ip address
!
interface Ethernet0/0.17
 encapsulation dot1Q 17
 ip address 125.1.17.7 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode
!
!
router ospf 1
 router-id 120.1.7.7
 redistribute bgp 700 subnets
!
router bgp 700
 neighbor 125.1.17.1 remote-as 100
 !
 address-family ipv4
  redistribute ospf 1 match internal external 1 external 2
  neighbor 125.1.17.1 activate
 !
 address-family ipv4 multicast
  network 125.1.79.0 mask 255.255.255.0
  redistribute ospf 1 match internal external 1 external 2
```

Juniper JunOS

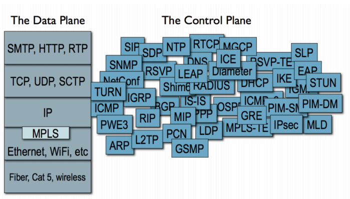
```
interfaces {
   so-0/0/0 {
        unit 0 {
            family inet {
                address 10.12.1.2/24;
            }
            family mpls;
        }
    }
   ge-0/1/0 {
        vlan-tagging;
        unit 0 {
            vlan-id 100;
            family inet {
                address 10.108.1.1/24;
            }
            family mpls;
        }
        unit 1 {
            vlan-id 200;
            family inet {
                address 10.208.1.1/24;
            }
        }
    }
…
}
protocols {
    mpls {
        interface all;
    }
    bgp {
```

A single mistyped line is enough
to bring down the entire network

```
redistribute bgp 700 subnets
```

Anything else than 700 creates blackholes

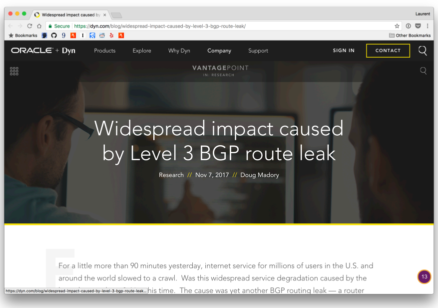
It's not only about the problem of configuring…
the level of complexity in networks is staggering



Source Mark Handley. Re-thinking the control architecture of the internet. Keynote talk. REARCH. December 2009.

Complexity + Low-level Management = **Problems**

November 2017



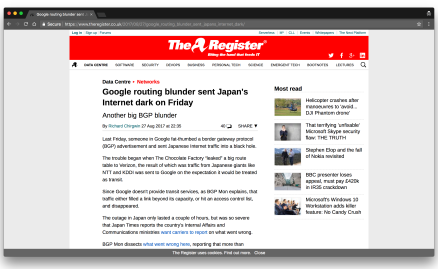
https://dyn.com/blog/widespread-impact-caused-by-level-3-bgp-route-leak/

For a little **more than 90 minutes** […],

Internet service for millions of users in the U.S. and around the **world slowed to a crawl**.

The cause was yet another BGP routing leak, a router misconfiguration directing Internet traffic from its intended path to somewhere else.

August 2017



https://www.theregister.co.uk/2017/08/27/google_routing_blunder_sent_japans_internet_dark/

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

[…] the result of which was **traffic from Japanese giants** like NTT and KDDI **was sent to Google on the expectation it would be treated as transit.**

The outage in Japan **only lasted a couple of hours**, but was so severe that […] the country's Internal Affairs and Communications ministries want carriers to report on what went wrong.

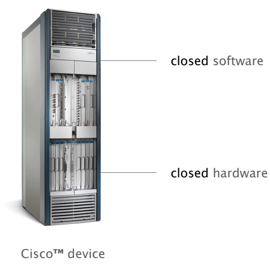
"Human factors are responsible
for 50% to 80% of network outages"

Juniper Networks, *What's Behind Network Downtime?*, 2008

"Cost per network outage
can be as high as 750 000$"

Smart Management for Robust Carrier Network Health
and Reduced TCO!, NANOG54, 2012

Solving this problem is hard because
network devices are completely locked down



Stage 2

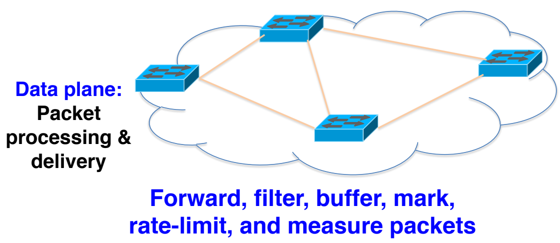
Software-Defined Networking

What is SDN and how does it help?

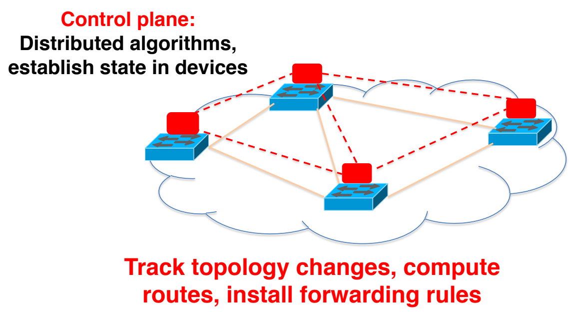
- SDN is a new approach to networking
  - Not about "architecture": IP, TCP, etc.
  - But about design of network control (routing, TE,…)
- SDN is predicated around two simple concepts
  - Separates the control-plane from the data-plane
  - Provides open API to directly access the data-plane
- While SDN doesn't do much, it enables *a lot*

Rethinking the "Division of Labor"

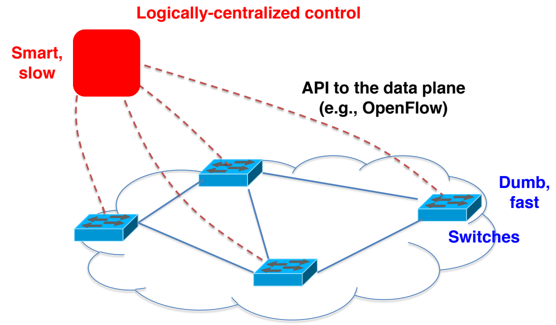
Traditional Computer Networks



Traditional Computer Networks

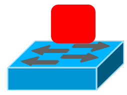


Software Defined Networking (SDN)



SDN advantages

- Simpler management
  - No need to “invert” control-plane operations
- Faster pace of innovation
  - Less dependence on vendors and standards
- Easier interoperability
  - Compatibility only in “wire” protocols
- Simpler, cheaper equipment
  - Minimal software



OpenFlow Networks

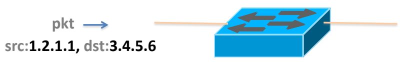
OpenFlow is an API to a switch flow table

- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

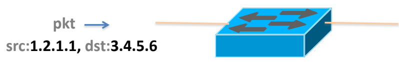
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

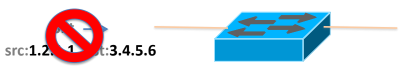
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

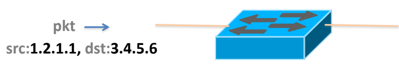
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

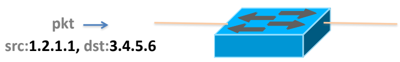
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

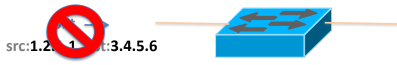
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets



10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow is an API to a switch flow table

- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

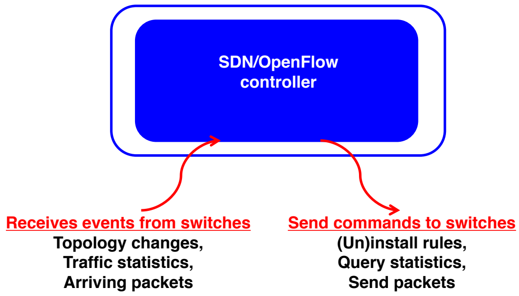


10. src=1.2.*.*, dest=3.4.5.* → drop
05. src = *.*.*.*, dest=3.4.*.* → forward(2)
01. src=10.1.2.3, dest=*.*.*.* → send to controller

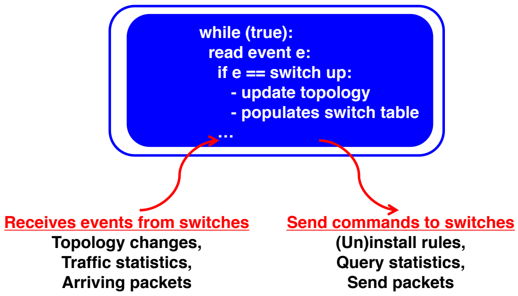
OpenFlow switches can emulate different kinds of boxes

- Router
  - Match: longest destination IP prefix
  - Action: forward out a link
- Switch
  - Match: destination MAC address
  - Action: forward or flood
- Firewall
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny
- NAT
  - Match: IP address and port
  - Action: rewrite address and port

Controller: Programmability



Controller: Programmability

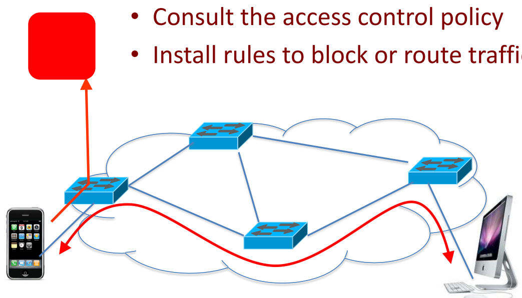


Example OpenFlow Applications

- **Dynamic access control**
- **Seamless mobility/migration**
- **Server load balancing**
- Network virtualization
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

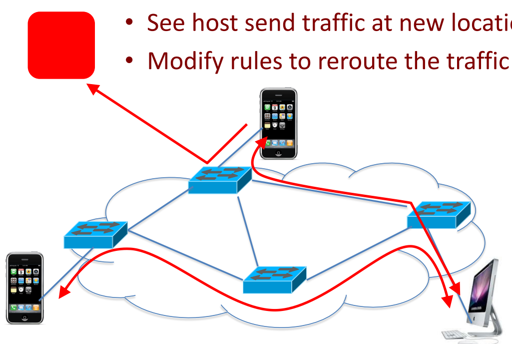
E.g.: Dynamic Access Control

- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic



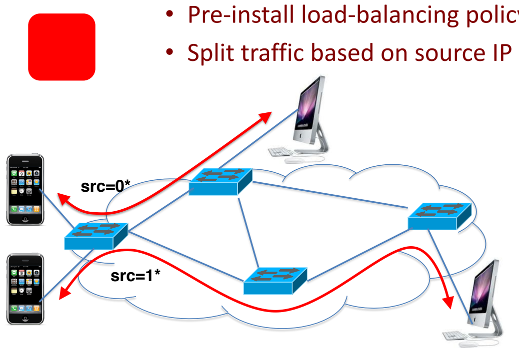
E.g.: Seamless Mobility/Migration

- See host send traffic at new location
- Modify rules to reroute the traffic



E.g.: Server Load Balancing

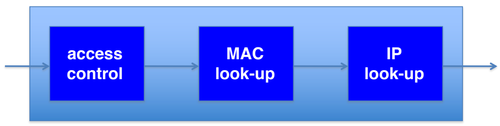
- Pre-install load-balancing policy
- Split traffic based on source IP



Challenges

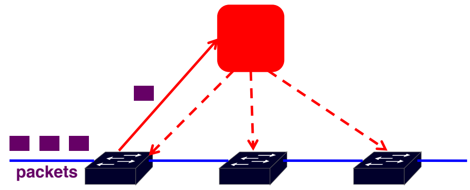
Heterogeneous Switches

- Number of packet-handling rules
- Range of matches and actions
- Multi-stage pipeline of packet processing
- Offload some control-plane functionality (?)

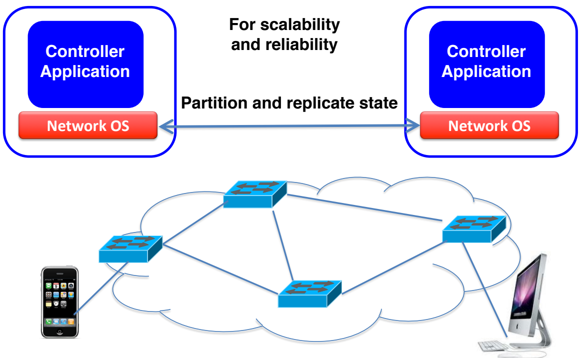


Controller Delay and Overhead

- Controller is much slower than the switch
- Processing packets leads to delay and overhead
- Need to keep most packets in the "fast path"



Distributed Controller

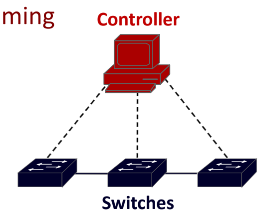


Testing and Debugging

- OpenFlow makes programming possible
  - Network-wide view at controller
  - Direct control over data plane
- Plenty of room for bugs
  - Still a complex, distributed system
- Need for testing techniques
  - Controller applications
  - Controller and switches
  - Rules installed in the switches

Programming Abstractions

- OpenFlow is a *low-level* API
  - Thin veneer on the underlying hardware
- Makes network programming possible, not easy!

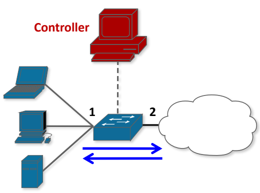


Example: Simple Repeater

Simple Repeater

```
def switch_join(switch):
  # Repeat Port 1 to Port 2
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)

  # Repeat Port 2 to Port 1
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
```



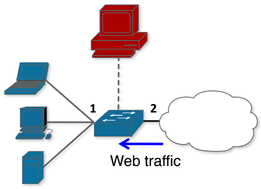
When a switch joins the network, install two forwarding rules.

Example: Web Traffic Monitor

Monitor "port 80" traffic

```
def switch_join(switch):
  # Web traffic from Internet
  p = {inport:2,tp_src:80}
  install(switch, p, DEFAULT, [])
  query_stats(switch, p)

def stats_in(switch, p, bytes, …)
  print bytes
  sleep(30)
  query_stats(switch, p)
```



When a switch joins the network, install one monitoring rule.

Composition: Repeater + Monitor

Repeater + Monitor

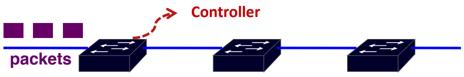
```
def switch_join(switch):
  pat1 = {inport:1}
  pat2 = {inport:2}
  pat2web = {in_port:2, tp_src:80}
  install(switch, pat1, DEFAULT, None, [forward(2)])
  install(switch, pat2web, HIGH, None, [forward(1)])
  install(switch, pat2, DEFAULT, None, [forward(1)])
  query_stats(switch, pat2web)

def stats_in(switch, xid, pattern, packets, bytes):
  print bytes
  sleep(30)
  query_stats(switch, pattern)
```

Must think about both tasks at the same time.

Asynchrony: Switch-Controller Delays

- Common OpenFlow programming idiom
  - First packet of a flow goes to the controller
  - Controller installs rules to handle remaining packets



- What if more packets arrive before rules installed?
  - Multiple packets of a flow reach the controller
- What if rules along a path installed out of order?
  - Packets reach intermediate switch before rules do

Must think about all possible event orderings.

Better: Increase the level of abstraction

- Separate reading from writing
  - Reading: specify queries on network state
  - Writing: specify forwarding policies
- Compose multiple tasks
  - Write each task once, and combine with others
- Prevent race conditions
  - Automatically apply forwarding policy to extra packets
- See http://frenetic-lang.org/

Stage 3

Deep Network Programability

Pinky — Gee, Brain, did OpenFlow take over the world?

The Brain — Well… no.

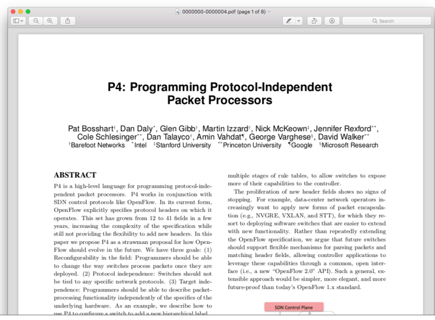
OpenFlow is not all roses

The protocol is too complex (12 fields in OF 1.0 to 41 in 1.5)
switches must support complicated parsers and pipelines

The specification itself keeps getting more complex
extra features make the software agent more complicated

consequences — Switches vendor end up implementing parts of the spec.
which breaks the abstraction of one API to *rule-them-all*

Enters… Protocol Independent Switch Architecture and P4


0000000-0000004.pdf (page 1 of 8)

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard†, Nick McKeown‡, Jennifer Rexford**, Cole Schlesinger**, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**
†Barefoot Networks *Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research

ABSTRACT

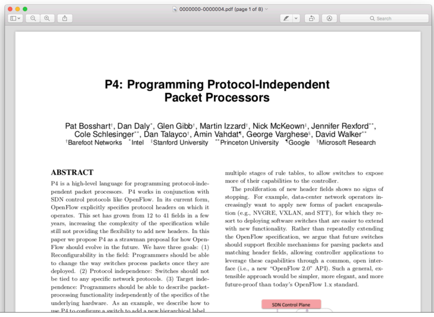
P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

SDN Control Plane

Enters… Protocol Independent Switch Architecture and P4


0000000-0000004.pdf (page 1 of 8)

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard†, Nick McKeown‡, Jennifer Rexford**, Cole Schlesinger**, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**
†Barefoot Networks *Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research

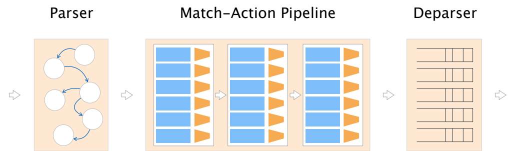
ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label

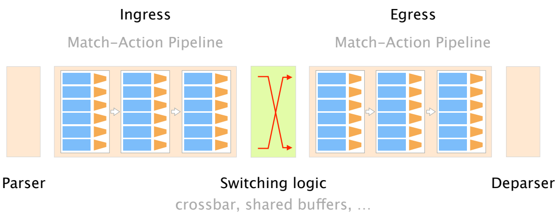
multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

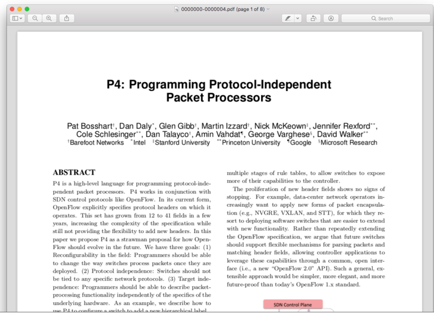
Protocol Independent Switch Architecture (PISA) for high-speed programmable packet forwarding



A slightly more accurate architecture



Enters… Protocol Independent Switch Architecture and P4


0000000-0000004.pdf (page 1 of 8)

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard†, Nick McKeown‡, Jennifer Rexford**, Cole Schlesinger**, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**
†Barefoot Networks *Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research

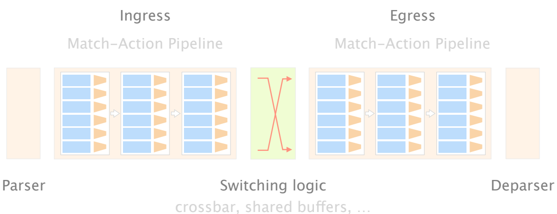
ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label

multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

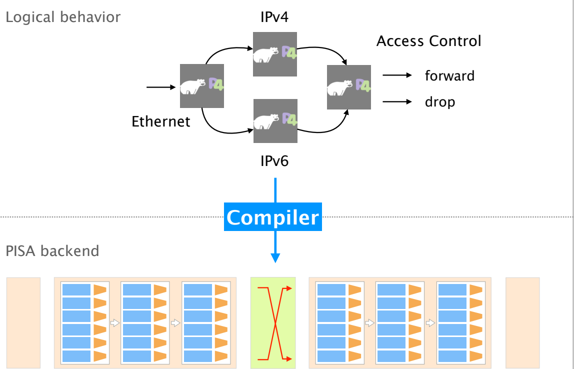
The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

By default,
PISA doesn't do anything, it's just an "architecture"

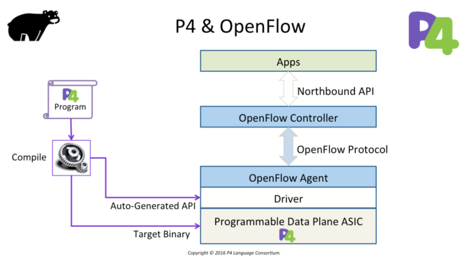


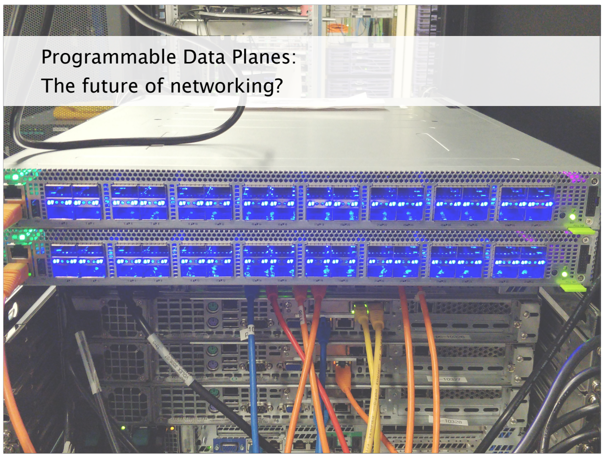
P4 is a domain-specific language which describes how a PISA architecture should process packets

https://p4.org

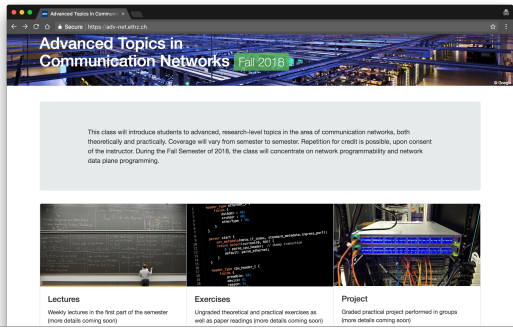


PISA + P4 is strictly more general OpenFlow





If you are interested, consider taking
Advanced Topics in Communication Networks [adv-net.ethz.ch]



Communication Networks
Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
May 13 2019