Communication Networks

Prof. Laurent Vanbever

Communication Networks

Spring 2017

Laurent Vanbever
www.vanbever.eu

ETH Zürich (D-ITET)
June 1 2017

Material inspired from Scott Shenker & Jennifer Rexford

Last Monday on
Communication Networks

DNS

Web

google.ch ⟷ 172.217.16.131

http://www.google.ch

DNS

Web

google.ch ⟷ 172.217.16.131

The DNS system is a distributed database
which enables to resolve a name into an IP address

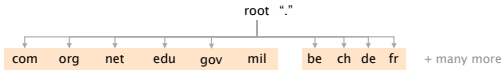
name	DNS ⟷	IP address
www.ethz.ch		129.132.19.216

To scale,
DNS adopt three intertwined hierarchies

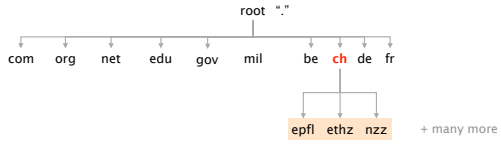
naming structure	addresses are hierarchical www.ee.ethz.ch
management	hierarchy of authority over names
infrastructure	hierarchy of DNS servers

naming structure	addresses are hierarchical www.ee.ethz.ch

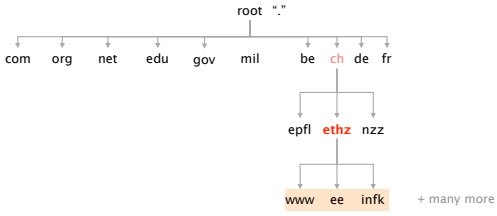
Top Level Domain (TLDs) sit at the top



Domains are subtrees



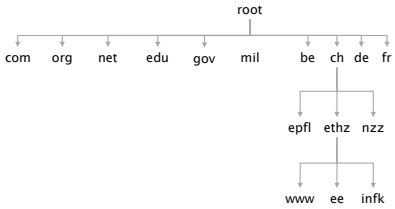
A name, *e.g.* ee.ethz.ch, represents
a leaf-to-root path in the hierarchy



management

hierarchy of authority
over names

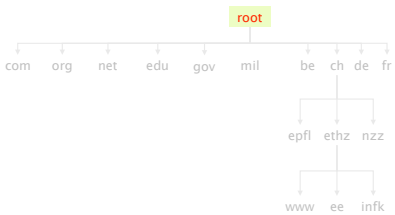
The DNS system is
hierarchically administered



infrastructure

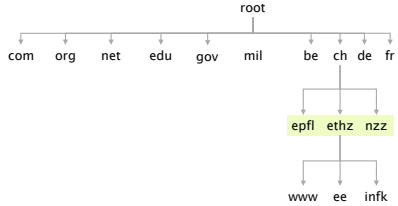
hierarchy of DNS servers

13 root servers (managed professionally)
serve as root (*)



(*) see http://www.root-servers.org/

The bottom (and bulk) of the hierarchy is
managed by Internet Service Provider or locally



Every server knows the address of the root servers (*)
required for bootstrapping the systems

(*) see https://www.internic.net/domain/named.root

From there on,
each server knows the address of all children

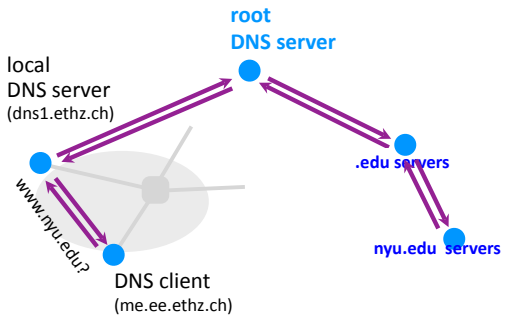
Using DNS relies on two components

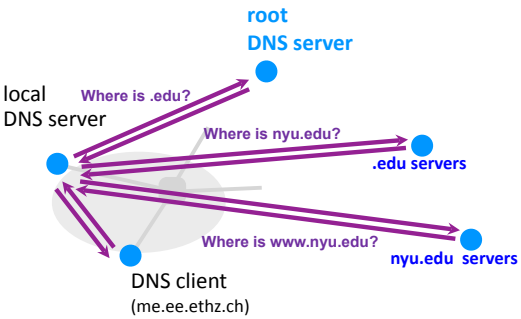
trigger resolution process
send request to local DNS server

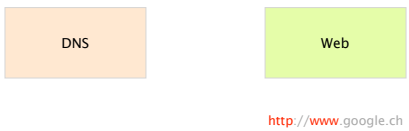
usually, near the endhosts
configured statically (resolv.conf)
or dynamically (DHCP)

Records	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

DNS resolution can either be
recursive or iterative







http://www.google.ch

The WWW is made of
three key components

Infrastructure	Content	Implementation
Clients/Browser Servers Proxies	Objects files, pictures, videos, … *organized* in Web sites a collection of objects	URL: name content HTTP: transport content

We focused on
its implementation

Infrastructure	Content	Implementation
Clients/Browser Servers Proxies	Objects files, pictures, videos, … *organized* in Web sites a collection of objects	URL: name content HTTP: transport content

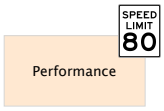
Infrastructure	Content	Implementation
Clients/Browser Servers Proxies	Objects files, pictures, videos, … *organized* in Web sites a collection of objects	URL: name content **HTTP: transport content**

HTTP is a rather simple
synchronous request/reply protocol

HTTP is layered over a bidirectional byte stream
almost always TCP

HTTP is text-based (ASCII)
human readable, easy to reason about

HTTP is stateless
it maintains *no info* about past client requests



Today on
Communication Networks

ICMP	NAT
Network Control Messages	Network Address Translation
its use for discovery	its use for sharing IPs

+ a little bit of SDN and course recap.

ICMP	NAT
Network Control Messages	
its use for discovery	

What Errors Might A Router See?

- Dead-end: No route to destination
- Sign of a loop: TTL expires
- Can't physically forward: packet too big
  - And has DF flag set
- Can't keep up with traffic: buffer overflowing
- Header corruption or ill-formed packets
- ….

What should network tell host about?

- No route to destination?
  - Host can't detect or fix routing failure.
- TTL expires?
  - Host can't detect or fix routing loop.
- Packet too big (with DF set)?
  - Host can adjust packet size, but can't tell difference between congestion drops and MTU drops
- Buffer overflowing?
  - Transport congestion control can detect/deal with this
- Header corruption or ill-formed packets?
  - Host can't fix corruption, but can fix formatting errors

Router Response to Problems?

- Router doesn't really need to respond
  - Best effort means never having to say you're sorry
  - So, IP could conceivably just silently drop packets
- Network is already trying its best
  - Routing is already trying to avoid loops/dead-ends
  - Network can't reduce packet size (in DF packets)
  - Network can't reduce load, nor fix format problems
- What more can/should it do?

Error Reporting Helps Diagnosis

- Silent failures are **really hard to diagnose**
- IP includes feedback mechanism for network problems, so they don't go undetected
- Internet Control Message Protocol (ICMP)
- The Internet "print" statement
- Runs on IP, but viewed as *integral* part of IP

Internet Control Message Protocol

- Triggered when IP packet encounters a problem
  - E.g., **Time Exceeded** or **Destination Unreachable**
- ICMP packet sent back to the source IP address
  - Includes the error information (e.g., type and code)
  - IP header plus 8+ byte *excerpt* from original packet
- Source host receives the ICMP packet
  - Inspects *excerpt* (e.g., protocol/ports) to identify socket
- **Exception**: not sent if problem packet is ICMP
  - And just for fragment 0 of a group of fragments

Types of Control Messages

- **Need Fragmentation**
  - IP packet too large for link layer, DF set
- **TTL Expired**
  - Decremented at each hop; generated if ⇒ 0
- **Unreachable**
  - Subtypes: network / host / port
    - (who generates Port Unreachable?)
- **Source Quench**
  - Old-style signal asking sender to slow down
- **Redirect**
  - Tells source to use a different local router

Using ICMP

- ICMP intended to tell host about network problems
  - **Diagnosis**
  - Won't say more about this….
- Can exploit ICMP to elicit network information
  - **Discovery**
  - Will focus on this….

Discovering Network Path Properties

- *PMTU Discovery*: Largest packet that can go through the network w/o needing fragmentation
  - Most efficient size to use
  - (Plus fragmentation can amplify loss)
- *Traceroute:*
  - What is the series of routers that a packet traverses as it travels through the network?
- *Ping:*
  - Simple RTT measurements

Ping: Echo and Reply

- ICMP includes simple "echo" functionality
  - Sending node sends an ICMP Echo Request message
  - Receiving node sends an ICMP Echo Reply
- Ping tool
  - Tests connectivity with a remote host
  - … by sending regularly spaced Echo Request
  - … and measuring delay until receiving replies

Path MTU Discovery

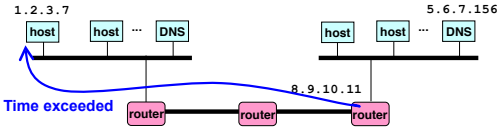
- MTU = Maximum Transmission Unit
  - Largest IP packet that a link supports
- Path MTU (PMTU) = minimum end-to-end MTU
  - Must keep datagrams no larger to avoid fragmentation
- How does the sender know the PMTU is?
- Strategy (RFC 1191):
  - **Try** a desired value
  - Set **DF** to prevent fragmentation
  - Upon receiving **Need Fragmentation** ICMP …
    - … oops, that didn't work, try a smaller value

Issues with Path MTU Discovery

- What set of values should the sender try?
  - Usual strategy: work through "likely suspects"
  - E.g., 4352 (FDDI), 1500 (Ethernet), 1480 (IP-in-IP over Ethernet), 296 (some modems)
- What if the PMTU changes? (how could it?)
  - Sender will immediately see *reductions* in PMTU (how?)
  - Sender can periodically try larger values
- What if **Needs Fragmentation** ICMP is lost?
  - Retransmission will elicit another one
- How can **The Whole Thing Fail**?
  - "PMTU **Black Holes**": routers that don't send the ICMP

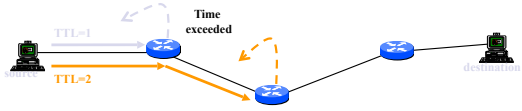
Discovering Routing via *Time Exceeded*

- Host sends an IP packet
  - Each router decrements the time-to-live field
- If **TTL** reaches 0
  - Router sends **Time Exceeded** ICMP back to the source
  - Message identifies router sending it
    - Since ICMP is sent using IP, it's just the IP source address
    - And can use PTR record to find name of router



Traceroute: Exploiting *Time Exceeded*

- Time-To-Live field in IP packet header
  - Source sends a packet with TTL ranging from ***1*** to ***n***
  - Each router along the path decrements the TTL
  - "TTL exceeded" sent when TTL reaches *0*
- *Traceroute* tool exploits this TTL behavior



Send packets with TTL=1, 2, …
and record source of *Time Exceeded* message

ICMP

NAT

Network Address Translation

its use for sharing IPs

Sharing Single Address Across Hosts

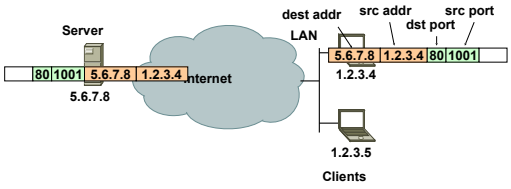
- Network Address Translation (NAT) enables many hosts to share a single address
  - Uses port numbers (fields in transport layer)
- Was thought to be an architectural abomination when first proposed, but it:
  - Probably saved us from address exhaustion
  - And reflects a modern design paradigm (indirection)

Special-Purpose Address Blocks

- Limited broadcast
  - Sent to every host attached to the local network
  - Block: **255.255.255.255/32**
- Loopback
  - Address blocks that refer to the local machine
  - Block: **127.0.0.0/8**
  - Usually only **127.0.0.1/32** is used
- Link-local
  - By agreement, not forwarded by any router
  - Used for single-link communication only
  - Intent: autoconfiguration (especially when *DHCP* fails)
  - Block: **169.254.0.0/16**
- Private addresses
  - By agreement, not routed in the public Internet
  - For networks not meant for general Internet connectivity
  - Blocks: **10.0.0.0/8**, **172.16.0.0/12**, **192.168.0.0/16**

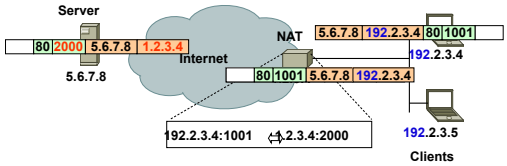
Network Address Translation (NAT)

Before NAT…every machine connected to Internet had unique IP address



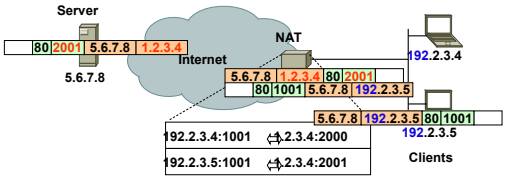
NAT (cont'd)

- Assign addresses to machines behind same NAT
  - Can be any private address range
  - e.g. **192.168.0.0/16**
- Use **port numbers** to multiplex single address



NAT (cont'd)

- Assign addresses to machines behind same NAT
  - Usually in address block **192.168.0.0/16**
- Use port numbers to multiplex single address



NAT: Early Example of "Middlebox"

- Boxes stuck into network to delivery functionality
  - NATs, Firewalls,….
- Don't fit into architecture, violate E2E principle
- But a very handy way to inject functionality that:
  - Does not require end host changes or cooperation
  - **Is under operator control (e.g., security)**
- An interesting architectural challenge:
  - How to incorporate middleboxes into architecture

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Software-Defined Networking & Research outlook

Laurent Vanbever

based on original slides by Prof. Jennifer Rexford & Prof. Scott Shenker

Networks are Hard to Manage

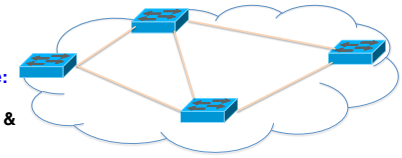
- Operating a network is expensive
  - More than half the cost of a network
  - Yet, operator error causes most outages
- Buggy software in the equipment
  - Routers with 20+ million lines of code
  - Cascading failures, vulnerabilities, etc.
- The network is "in the way"
  - Especially a problem in data centers
  - … and home networks

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

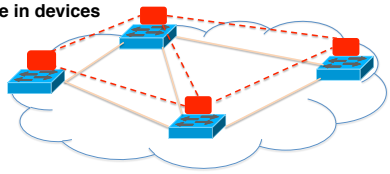


Data plane: Packet processing & delivery

Forward, filter, buffer, mark, rate-limit, and measure packets

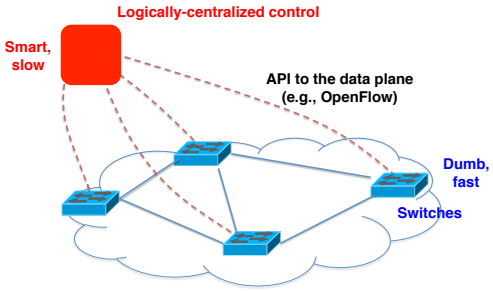
Traditional Computer Networks

Control plane: Distributed algorithms, establish state in devices



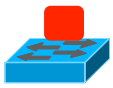
Track topology changes, compute routes, install forwarding rules

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

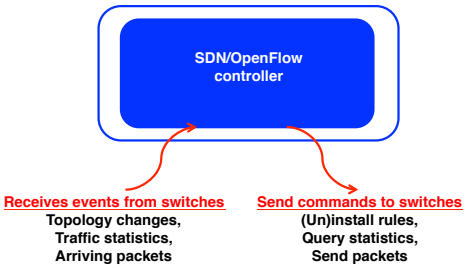
- Simple packet-handling rules
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

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

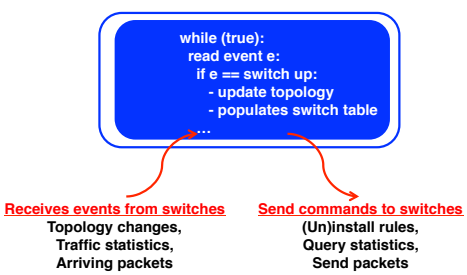
OpenFlow switches can emulate different kinds of boxes

- 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

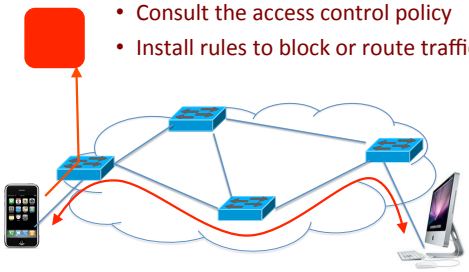


Example OpenFlow Applications

- **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

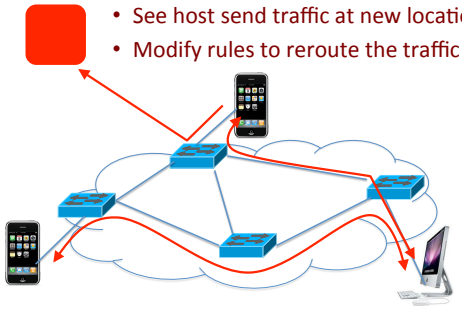
E.g.: Dynamic Access Control

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



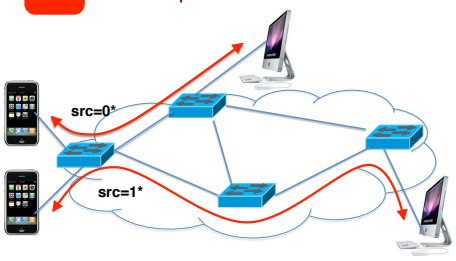
E.g.: Seamless Mobility/Migration

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



E.g.: Server Load Balancing

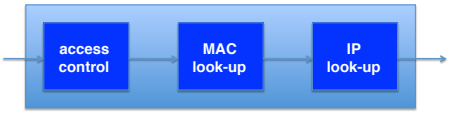
- Pre-install load-balancing policy
- Split traffic based on source IP



Challenges

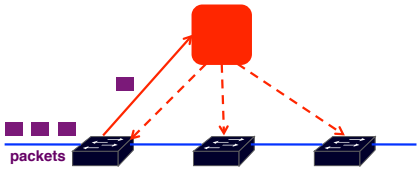
Heterogeneous Switches

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

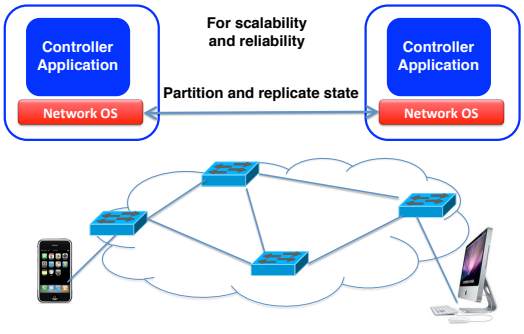


Controller Delay and Overhead

- 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

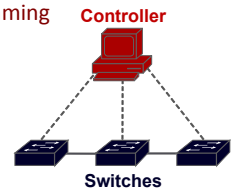


Testing and Debugging

- 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!

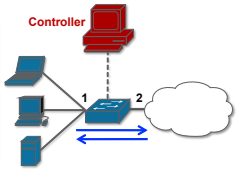


Example: Simple Repeater

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)
```



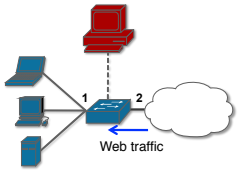
When a switch joins the network, install two forwarding rules.

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

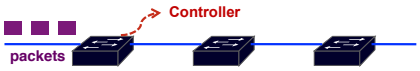
```
def switch_join(switch):
  pat1 = {inport:1}
  pat2 = {inport:2}
  pat2web = {in_port:2, tp_src:80}
  install(switch, pat1, DEFAULT, None, [forward(2)])
  install(switch, pat2web, HIGH, None, [forward(1)])
  install(switch, pat2, DEFAULT, None, [forward(1)])
  query_stats(switch, pat2web)

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/

Current research directions

- Bring SDN to the Internet
- Enable SDN in existing networks
- Boost the performance of existing networks using SDN
- Verify controller programs and interactions
- Improve network monitoring
- Improve network security and anonymity
- … and many more!

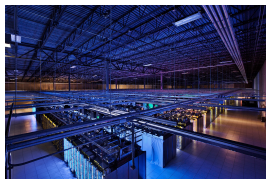
Conclusions

- SDN is exciting
  - Enables innovation
  - Simplifies management
  - Rethinks networking from the ground-up
- Significant momentum
  - In both research and industry
  - Size of the SDN market already several billion $$
- **Great** research opportunity
  - Practical impact on future networks practices
  - Placing network on a strong foundation

Knowledge

Understand how the Internet works and why

from your network plug…



…to Google's data-center

List any

technologies, principles, applications…

used after typing in:

> www.google.ch

and pressing enter in your browser

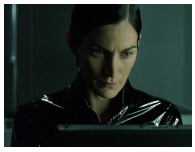
Insight

Key concepts and problems in Networking

Naming Layering Routing Reliability Sharing

Skill

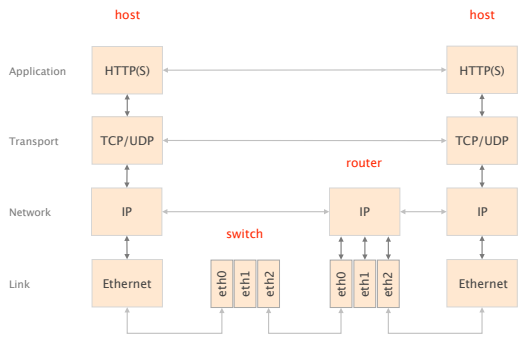
Build, operate and configure networks



Trinity using a port scanner (nmap) in Matrix Reloaded™

The Internet is organized as layers, providing a set of services

	layer	service provided
L5	Application	network access
L4	Transport	end-to-end delivery (reliable or not)
L3	Network	global best-effort delivery
L2	Link	local best-effort delivery
L1	Physical	physical transfer of bits



We started with the fundamentals of routing and reliable transport

	Application	network access
L4	Transport	end-to-end delivery (reliable or not)
L3	Network	global best-effort delivery
	Link	local best-effort delivery
	Physical	physical transfer of bits

We saw **three ways to compute** valid routing state

	Intuition	Example
#1	Use tree-like topologies	Spanning-tree
#2	Rely on a global network view	Link-State SDN
#3	Rely on distributed computation	Distance-Vector BGP

We saw **how to design a reliable transport protocol**

goals

correctness	ensure data is delivered, in order, and untouched
timeliness	minimize time until data is transferred
efficiency	optimal use of bandwidth
fairness	play well with other concurrent communications

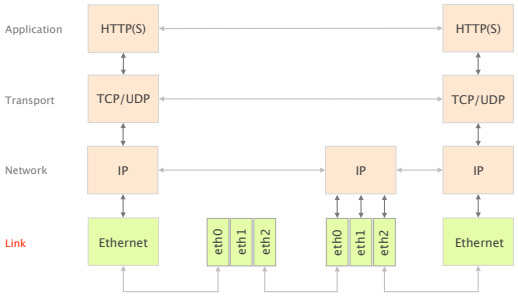
In each case, we explored the rationale behind each protocol and why they came to be

Why did the protocols end up looking like this?
minimum set of features required

What tradeoffs do they achieve?
efficiency, cost,…

When is one design more adapted than another?
packet switching *vs* circuit switching, DV *vs* LS,…

We then climbed up the layers, starting from layer 2

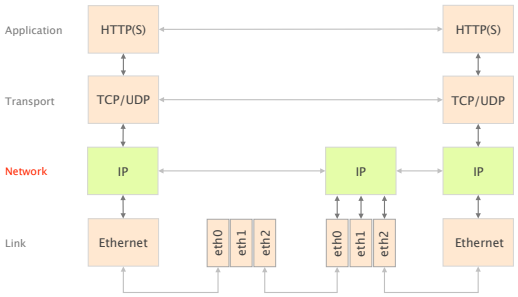


Communication Networks

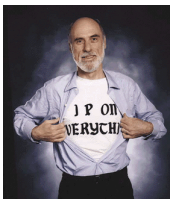
Part 2: The Link Layer

#1	What is a link?
#2	How do we identify link adapters?
#3	How do we share a network medium?
#4	What is Ethernet?
#5	How do we interconnect segments at the link layer?

We then spent multiple weeks on layer 3



Internet Protocol and Forwarding



source: Boardwatch Magazine

1. **IP addresses**
   use, structure, allocation
2. **IP forwarding**
   longest prefix match rule
3. **IP header**
   IPv4 and IPv6, wire format

Internet routing

from here to there, and back

1. **Intra-domain routing**
   Link-state protocols
   Distance-vector protocols
2. **Inter-domain routing**
   Path-vector protocols

Border Gateway Protocol
policies and more

1 **BGP Policies**
Follow the Money

2 **Protocol**
How does it work?

3 **Problems**
security, performance, …

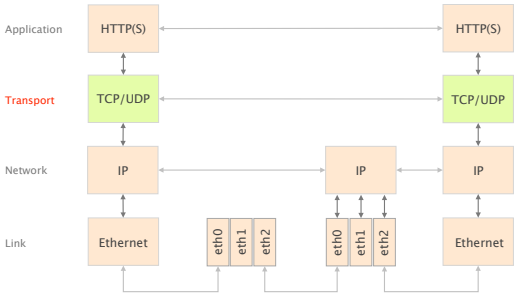
Routing security
attacks & mitigation

intra-domain routing	inter-domain routing
insider	in/outsider

BGP (lack of) security:
problems & solutions

#1	BGP does not validate the origin of advertisements
#2	BGP does not validate the content of advertisements
#3	Proposed Enhancements
#4	What about the data plane?
#5	What's the Internet to do anyway?

4 = 3+1



Transport Protocols: UDP & TCP
The requirements

Data delivering, to the *correct* application
- IP just points towards next protocol
- *Transport needs to demultiplex incoming data (ports)*

Files or bytestreams abstractions for the applications
- Network deals with packets
- *Transport layer needs to translate between them*

Reliable transfer (if needed)
Not overloading the receiver
Not overloading the network

Transport Protocols: UDP & TCP
The implementation

Demultiplexing: identifier for application process
- Going from host-to-host (IP) to process-to-process

Translating between bytestreams and packets:
- Do segmentation and reassembly

Reliability: ACKs and all that stuff
Corruption: Checksum
Not overloading receiver: "Flow Control"
- Limit data in receiver's buffer

Not overloading network: "Congestion Control"

We then looked at Congestion Control
and how it solves three fundamental problems

#1	bandwidth estimation	How to adjust the bandwidth of a single flow to the bottleneck bandwidth? could be 1 Mbps or 1 Gbps…
#2	bandwidth adaptation	How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
#3	fairness	How to share bandwidth "fairly" among flows, without overloading the network

… by combining two key mechanisms

detecting congestion	reacting to congestion

We finally looked at
what's running on top of all this …

DNS

google.ch ⟷ 172.217.16.131

Web

http://www.google.ch

… and filled-up some holes
with 2 helpers protocols

ICMP

Network Control Messages

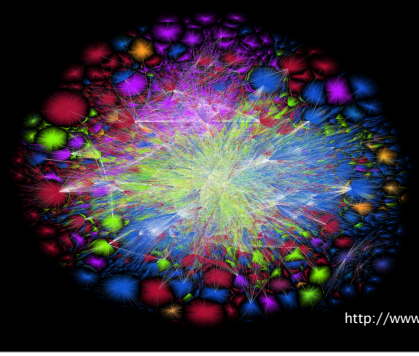
its use for discovery

NAT

Network Address Translation

its use for sharing IPs

Now you (better) understand this!



http://www.opte.org

We've opportunities for both
Master and Semester theses

soon available on our
upcoming website:

nsg.ee.ethz.ch (coming soon)

for now, shoot us an email

Communication Networks

Spring 2017

ETH

Laurent Vanbever
www.vanbever.eu

ETH Zürich (D-ITET)
June 1 2017