We'll study e-mail from three different perspectives

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

A header, 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

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

Ivanbever @ ethz.ch

local mailbox  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
This week on Communication Networks

programmable networks IPv6 next generation of Internet addressing next generation of network devices

The long way from…

World population: 7.5 billion
~0.6 IPv4 addresses per person

...to...

Average # of atoms in a human: \(6.1 \times 10^{27}\)
~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
1993 experts warn that IPv4 addresses might run out
1994 “Address Allocation for Private Internets”
3 reserved IPv4 blocks for private networks
Hosts in private IP space are unreachable from Internet
late 1990s “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 adoption of IPv6
Did not happen...
2008 It is possible to resolve domain names using IPv6 only

IPv6 originally appeared in 1998
i.e. more than 20 years ago

... 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


Not all countries are equivalent though

The darker the green, the larger the deployment


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

Most of IPv6 new features were back-ported to IPv4

There is no obvious advantage in using IPv6

Deploying IPv6 requires every device to support it

All routers, middleboxes, end hosts, applications, …

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

R

Local Network 1.2.3.0/24

1.2.3.4

1.2.3.5

Server

5.6.7.8

port 80

R

5.6.7.8:800

12.3.4.2001

12.3.4.5

Internet

R

The Internet with NAT

The port numbers are used to multiplex single addresses

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

<table>
<thead>
<tr>
<th>Notation</th>
<th>8 groups of 16 bits each separated by colons (:)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Each group is written as four hexadecimal digits</td>
</tr>
</tbody>
</table>

**Simplification**

<table>
<thead>
<tr>
<th>Leading zeros in any group are removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>One section of zeros is replaced by a double colon (:)</td>
</tr>
<tr>
<td>Normally the longest section</td>
</tr>
</tbody>
</table>

**Examples**

- 1080::0:8:800:200C:417A
- FF01::101
- ::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

Global unicast addresses are hierarchically allocated

similar to global IPv4 addresses

<table>
<thead>
<tr>
<th>128 bits</th>
<th>N bits</th>
<th>M bits</th>
<th>128-N-M bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>global routing prefix</td>
<td>subnet ID</td>
<td>Interface ID</td>
<td></td>
</tr>
</tbody>
</table>

Identifies the ISP responsible for the address
Usually 64 bits
Based on the MAC address
A subnet in this ISP or a customer of the ISP

Allocation of IPv6 (global unicast) addresses

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

<table>
<thead>
<tr>
<th>128 bits</th>
<th>10 bits</th>
<th>54 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE80</td>
<td>0000...0000</td>
<td>Interface ID</td>
<td></td>
</tr>
</tbody>
</table>

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**

0000:0000:0000:0000:0000:0000:0000:0000
Used as src address if no IPv6 address available

**Loopback address**

0000:0000:0000:0000:0000:0000:0000:0001
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 addresses identify a group of receivers/interfaces

![Multicast addresses diagram](image)

- Identifies a set of interfaces
- Packets are delivered to all interfaces

Some multicast addresses are well-known and used for auto-discovery, bootstrapping, 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

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

The IPv6 packet header format

![IPv6 packet header diagram](image)

- 32 bits
- Version
- Traffic Class
- Flow Label
- Payload Length
- Next Header
- Hop Limit
- Source IPv6 address
- Destination IPv6 address

Extension header example: ICMPv6

![Extension header diagram](image)

- Similar functions than IPv4 ICMP
- The next header field indicates the type of the extension header
ICMPv6 can be used for neighbor discovery
replacement for IPv4’s ARP

First step: neighbor solicitation

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>135</td>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Target IPv6 address

Second step: neighbor advertisement

Is a router?

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Target IPv6 address

Requested link-layer address

Answer to neighbor solicitation?

Target link-layer address

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::M64(800:200C:417A)
M64: 64-bit representation of the MAC address

Neighbor solicitation for FE80: M64(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::M64(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

Data Link (Ethernet)
IPv6
IPv4
TCP
UDP
Application
Over the years, a lot of transition mechanisms were developed. Some of these include:

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

... etc.

Tunnel IPv6 packets over static IPv4 links (6in4)

IPv4 header

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Version</th>
<th>Src/dst IPs</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>6 (TCP)</td>
<td>IPv6</td>
<td>6 (IPv6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IPv6 payload

<table>
<thead>
<tr>
<th>Application</th>
<th>TCP header and ports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not available</td>
</tr>
</tbody>
</table>

IPv6 @ home (Swisscom Internet access box)

You will be assigned an IPv4 and IPv6 address.

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

Network programmability is attracting tremendous industry interest (and money).

Network programmability is getting traction in many academic communities.

Networking Systems
Security
Distributed Algorithms
SIGCOMM
SIGDI
HotNets
CoNEXT
...

Security
POPL
OOPSLA
CCS
Usenix
SIGCOMM
OSDI
SOSP
...

PLDI
NDSS
POPL
OOPSLA
CCS
Usenix
SIGCOMM
OSDI
SOSP
Why? It’s really a story in 3 stages

Stage 1
The network management crisis

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

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

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

Adapt C so that the network follows the new behavior
The level of complexity in networks is staggering. Configuring each element is often done manually, using arcane low-level, vendor-specific “languages.”

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

Complexity + Low-level Management = Problems


November 2017

Widespread impact caused by Level 3 BGP route leak

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

August 2017

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

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 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.

For 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.

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.

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

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

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

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)**

- **Smart, slow**

**API to the data plane**
(e.g., OpenFlow)

- **Dumb, fast**

**OpenFlow Networks**

**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 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

**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

**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
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

- Receives events from switches: Topology changes, Traffic statistics, Arriving packets
- Sends commands to switches: (Un)install rules, Query statistics, Send packets

Controller: Programmability

while (true):
    read event e:
    if e == switch up:
        - update topology
        - populate switch table

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”

**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**
```python
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

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

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

Composition: Repeater + Monitor

```python
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)
```

Repeater + Monitor
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


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
- **Switch vendor end up implementing parts of the spec.**
  - Which breaks the abstraction of one API to rule-them-all

Stage 3
Deep Network Programability

Enters... Protocol Independent Switch Architecture and P4
Protocol Independent Switch Architecture (PISA) for high-speed programmable packet forwarding

A slightly more accurate architecture

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

PISA + P4 is strictly more general OpenFlow

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

https://p4.org
Programmable Data Planes: The future of networking?

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