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

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
Top Level Domain (TLDs) sit at the top

```
root "."
```

- com
- org
- net
- edu
- gov
- mil
- be
- ch
- de
- fr
- + many more

```
```

- ethz
- epfl
- nzz
- + many more

Domains are subtrees

```

```
```

- epfl
- ethz
- nzz
- + many more

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

```

```

- epfl
- ethz
- nzz
- www
- ee
- infk
- + many more

The DNS system is hierarchically administered

```

```

- epfl
- ethz
- nzz
- www
- ee
- infk

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

```

```

- epfl
- ethz
- nzz
- www
- ee
- infk

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

```

```

- epfl
- ethz
- nzz
- www
- ee
- infk

(*) see http://www.root-servers.org/
Every server knows the address of the root servers (*) required for bootstrapping the systems.

Using DNS relies on two components:
- Resolver software
- Local DNS server

Dynamically (DHCP), or configured statically (resolv.conf), the local DNS server is usually near the endhosts and triggers the resolution process.

DNS resolution can either be recursive or iterative.

Web DNS resolution example:
- DNS client (me.ee.ethz.ch)
- Root DNS server
- Local DNS server (dns1.ethz.ch)
- nyu.edu servers
- www.nyu.edu?
- .edu servers

Records

<table>
<thead>
<tr>
<th>Records</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>hostname</td>
<td>IP address</td>
</tr>
<tr>
<td>NS</td>
<td>domain</td>
<td>DNS server name</td>
</tr>
<tr>
<td>MX</td>
<td>domain</td>
<td>Mail server name</td>
</tr>
<tr>
<td>CNAME</td>
<td>alias</td>
<td>canonical name</td>
</tr>
<tr>
<td>PTR</td>
<td>IP address</td>
<td>corresponding hostname</td>
</tr>
</tbody>
</table>
The WWW is made of three key components

- **Infrastructure**
- **Content**
- **Implementation**

<table>
<thead>
<tr>
<th>Clients/Browser</th>
<th>Objects</th>
<th>URL: name content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servers</td>
<td></td>
<td>HTTP: transport content</td>
</tr>
<tr>
<td>Proxies</td>
<td>files, pictures, videos, ...</td>
<td>organized in Web sites</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a collection of objects</td>
</tr>
</tbody>
</table>

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**

- Video Streaming
- E-mail

HTTP-based

**We want the highest video quality**

(c) copyright 2008, Blender Foundation / www.bigbuckbunny.org, CC-BY 3.0

Without seeing this ...
Why should you care? Just look at this: video's share of global internet traffic

A naive approach: one-size-fits-all

In practice, things are slightly more complex

The three steps behind most contemporary solutions

- Encode video in multiple bitrates
- Replicate using a content delivery network
- Video player picks bitrate adaptively
- Estimate connection's available bandwidth
- Pick a bitrate ≤ available bandwidth

(Adapted from: Adaptive Streaming of Traditional and Omnidirectional Media, Begen & Timmerer, ACM SIGCOMM Tutorial, 2017)
Simple solution for encoding: use a “bitrate ladders”

<table>
<thead>
<tr>
<th>Bitrate (kbps)</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>300x400</td>
</tr>
<tr>
<td>391</td>
<td>390x400</td>
</tr>
<tr>
<td>400</td>
<td>360x400</td>
</tr>
<tr>
<td>510</td>
<td>510x400</td>
</tr>
<tr>
<td>600</td>
<td>510x400</td>
</tr>
<tr>
<td>700</td>
<td>600x800</td>
</tr>
<tr>
<td>800</td>
<td>720x500</td>
</tr>
<tr>
<td>900</td>
<td>720x500</td>
</tr>
<tr>
<td>1100</td>
<td>1080x720</td>
</tr>
<tr>
<td>1200</td>
<td>1280x720</td>
</tr>
<tr>
<td>1400</td>
<td>1440x900</td>
</tr>
</tbody>
</table>

Your player download “chunks” of video at different bitrates

Depending on your network connectivity, your player fetches chunks of different qualities

Your player gets metadata about chunks via “Manifest”

Play Connect: Starting from a Greenfield (a mostly Layer 0 talk)

Storage Appliance
- 500-600 Tbps
- ~500 watts
- 2x 10G ports
- 288 TB of storage
- 2x 100 ports
- 256Gbps delivery

Flash Appliance
- 1U
- ~175 watts
- 2x 40G ports
- 2x 40Gbps delivery

Encoding
Application
Adaptation

- NETFLIX
- Open Connect
- Starting from a Greenfield
- (a mostly Layer 0 talk)

Dave Temkin
06/01/2015
1. INTRODUCTION

HTTP-based Video Streaming, Video Rate Adaptation Algorithms

Existing ABR algorithms face a significant challenge in estimating future capacity: capacity can vary widely over time, resulting in a phenomenon commonly observed in commercial services. While ABR algorithms used by such services balance two objectives—minimizing the number of rebuffers and maximizing video quality—these objectives often lead to suboptimal performance on both metrics in order to give users a good viewing experience.

We start with a random sample of 300,000 Netflix sessions that shows that roughly 10% of sessions experience a throughput less than half of the 95th percentile throughput. Current literature presents two techniques to address this. One approach is to pick a video rate by estimating future capacity, whereas the other is to adaptively pick video rates. The former reduces the number of rebuffers while the latter improves video quality. However, using simple capacity estimation (based on immediate past throughput) is important during the startup phase, whereas using simple capacity estimation (based on immediate past throughput) is important during the startup phase, whereas using simple capacity estimation (based on immediate past throughput) is important during the startup phase.

To resolve this trade-off, we propose a buffer-based approach to the ABR problem. The basic idea is that for video streaming, there is a buffer in the hosting device, and it is filled by downloading chunks at different rates. When the buffer occupancy reaches a high level, the video player picks a bitrate adaptively based on the buffer state, and then downloads chunks at different rates. The video client—running on a home computer, TV, game console, web browser, DVD player, etc.—chooses a chunk download rate at the highest rate the network can deliver. This choice is made in a way such that downloading events are synchronized with playback.

As a result of this mechanism, the video starts playing within seconds. Each video is encoded using different bitrates, and the video client is able to select the best bitrate for the current network condition. The network is utilized efficiently, and the video quality is good. This approach reduces the number of rebuffers and improves video quality.

Evidence from a Large Video Streaming Service

“A random sample of 300,000 Netflix sessions shows that roughly 10% of sessions experience a median throughput less than half of the 95th percentile throughput.”

“20–30% of rebuffers are unnecessary.”
Capacity estimation

**Capacity (Mbps)**

**Time**

**Decide based on the buffer alone?**

**Buffer-based adaptation**

**Nearly full buffer ⇒ large rate**

**Buffer-based adaptation**

**Nearly empty buffer ⇒ small rate**

**Problem: startup phase?**

**Pick a rate based on immediate past throughput**

**Summary**

- Encode video in multiple bitrates
- Replicate using a content delivery network
- Video player picks bitrate adaptively
- Problem of active research interest, many competing algorithms and objectives

**We’ll study e-mail from three different perspectives**

**E-mail**

MX, SMTP, POP, IMAP

**Content**

**Infrastructure/Transmission**

Format: Header/Content
Encoding: MIME
SMTP: Simple Mail Transfer Protocol
Infrastructure mail servers

**Retrieval**

POP: Post Office Protocol
IMAP: Internet Message Access Protocol
An e-mail is composed of two parts:

- **Header**: in 7-bit U.S. ASCII text
- **Body**: also in 7-bit U.S. ASCII text

### 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

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

**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
MIME defines
- additional headers for the email body
- a set of content types and subtypes
- base64 to encode binary data in ASCII

MIME defines
- additional headers for the email body
- a set of content types and subtypes
  e.g. image with subtypes gif or jpeg
text with subtypes plain, html, and rich text
application with subtypes postscript or msword
multipart with subtypes mixed or alternative

MIME defines
- Content-Type: the type of data contained in the message
- Content-Transfer-Encoding: how the data are encoded

MIME relies on Base64 as binary-to-text encoding scheme

Relies on 64 characters out of the 128 ASCII characters
the most common and printable ones, i.e. A-Z, a-z, 0-9, +, /
Divides the bytes to be encoded into sequences of 3 bytes
each group of 3 bytes is then encoded using 4 characters
Uses padding if the last sequence is partially filled
i.e. if the sequence to be encoded is not a multiple of 3

The two most common types/subtypes for MIME are:
**multipart/mixed** and **multipart/alternative**

<table>
<thead>
<tr>
<th>Content-Type</th>
<th>indicates that the message contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>multipart/mixed</td>
<td>multiple independent parts</td>
</tr>
<tr>
<td>multipart/alternative</td>
<td>multiple representation of the same content</td>
</tr>
</tbody>
</table>

**Content-Type**

<table>
<thead>
<tr>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
<th>Value</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>31</td>
<td>z</td>
<td>62</td>
<td>@</td>
<td>63</td>
<td>\n</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>32</td>
<td>d</td>
<td>64</td>
<td>b</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>33</td>
<td>e</td>
<td>66</td>
<td>c</td>
<td>67</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>34</td>
<td>f</td>
<td>68</td>
<td>d</td>
<td>69</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
<td>35</td>
<td>g</td>
<td>70</td>
<td>e</td>
<td>71</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>f</td>
<td>36</td>
<td>h</td>
<td>72</td>
<td>f</td>
<td>73</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>37</td>
<td>i</td>
<td>74</td>
<td>g</td>
<td>75</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>h</td>
<td>38</td>
<td>j</td>
<td>76</td>
<td>h</td>
<td>77</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>i</td>
<td>39</td>
<td>k</td>
<td>78</td>
<td>i</td>
<td>79</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>j</td>
<td>40</td>
<td>l</td>
<td>80</td>
<td>j</td>
<td>81</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>k</td>
<td>41</td>
<td>m</td>
<td>82</td>
<td>k</td>
<td>83</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>l</td>
<td>42</td>
<td>n</td>
<td>84</td>
<td>l</td>
<td>85</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>m</td>
<td>43</td>
<td>o</td>
<td>86</td>
<td>m</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>n</td>
<td>44</td>
<td>p</td>
<td>88</td>
<td>n</td>
<td>89</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>45</td>
<td>q</td>
<td>90</td>
<td>o</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>p</td>
<td>46</td>
<td>r</td>
<td>92</td>
<td>p</td>
<td>93</td>
<td>4</td>
</tr>
</tbody>
</table>

Binary input

0x14fb9c03d97e

8-bits

00010100 11111011 10011110 00000001 11011001 01111110

6-bits

000101 001111 110110 010001 111101 011110

Decimal

5 15 46 28 0 61 37 62

base64

F p u c A 9 1 +
If the length of the input is not a multiple of three, Base64 uses "=" as padding character.

| Binary input | 0x14 |
| 8-bits | 00010100 |
| 6-bits | 000101000000 |
| Decimal | 5 0 |
| base64 | FA = = |

From: Laurent Vanbever <lvanbever@ethz.ch>
To: Tobias Buehler <buehlert@ethz.ch>
Subject: [comm-net] Final exam
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: multipart/mixed;
boundary="123boundary"

This is a multipart message in MIME format.

--123boundary
Content-Type: text/plain
Hi Tobias, Please find the exam enclosed. Laurent
--123boundary
Content-Type: application/pdf;
Content-Disposition: attachment;
filename="exam_2018.pdf"

base64 encoded data ......
.........................
......base64 encoded data

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

We can divide the e-mail infrastructure into five functions

Mail User Agent Use to read/write emails (mail client)
Mail Submission Agent Process email and forward to local MTA
Mail Transmission Agent Queues, receives, sends mail to other MTAs
Mail Delivery Agent Deliver email to user mailbox
Mail Retrieval Agent Fetches email from user mailbox

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

SMTP 3 digit response code comment
Success
2XX success
220 Service ready
250 Requested mail action completed
Input needed
3XX input needed
354 Start mail input
Transient error
4XX transient error
421 Service not available
450 Mailbox unavailable
451 Insufficient space
Permanent error
5XX permanent error
500 Syntax error
502 Unknown command
503 Bad sequence

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)
IMAP or POP is used to retrieve the e-mail by the recipient MUA. The sender MUA uses SMTP to transmit the e-mail first to a local MTA (e.g. mail.ethz.ch, gmail.com, hotmail.com).

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

Each SMTP server/MTA hop adds its identity to the e-mail header by prepending a "Received" entry.

E-mails typically go through at least 2 SMTP servers, but often way more.

Separate SMTP servers for separate functions: SPAM filtering, virus scanning, data leak prevention, etc.

Separate SMTP servers that redirect messages: e.g. from lvanbever@tik.ee.ethz.ch to lvanbever@ethz.ch.

Separate SMTP servers to handle mailing list: mail is delivered to the list server and then expanded.
As with most of the key Internet protocols, security is an afterthought.

SMTP-MSA
rely on TLS encryption
authentication required

In short, none of the addresses in an email are typically reliable.

And, as usual, multiple countermeasures have been proposed with various level of deployment success

Example:
Sender Policy Framework (SPF)
Enables a domain to explicitly authorize a set of hosts that are allowed to send emails using their domain names in "MAIL FROM".

How? using a DNS TXT resource record
look for "v=spf1" in the results of "dig TXT google.com"

* if you are interested, also check out Sender ID, DKIM, and DMARC.

POP is a simple protocol which was designed to support users with intermittent network connectivity

POP enables e-mail users to
- retrieve e-mails locally when connected
- view/manipulate e-mails when disconnected

and that's pretty much it…

Example
POP server
-OK POP3 server ready
user bob
-OK
pass hungry
-OK user successfully logged on
list
1 498
2 912
retr 1 <message 1 contents>
dele 1
retr 2 <message 1 contents>
dele 2
quit
-OK POP3 server signing off
Authorization phase
Clients declares username
password
Server answers +OK/-ERR

+OK POP3 server ready
user bob
OK
pass hungry
+OK user successfully logged on

+OK POP3 server ready
user bob
OK
pass hungry
+OK user successfully logged on

Transaction phase
list
retr
dele
quit

+OK POP3 server signing off

Transaction phase
list
get message numbers
retr
retrieve message X
dele
delete message X
quit
exit session

+OK POP3 server signing off

Not designed to keep messages on the server
designed to download messages to the client
Cannot deal with multiple mailboxes
designed to put incoming emails in one folder
Poor handling of multiple client access
while many (most?) users have now multiple devices

Unlike POP, Internet Message Access Protocol (IMAP) was designed with multiple clients in mind

Support multiple mailboxes and searches on the server
client can create, rename, move mailboxes & search on server
Access to individual MIME parts and partial fetch
client can download only the text content of an e-mail
Support multiple clients connected to one mailbox
server keep state about each message (e.g. read, replied to)

POP is heavily limited. Among others, it does not go well with multiple clients or always-on connectivity

Cannot deal with multiple mailboxes
designed to put incoming emails in one folder
Not designed to keep messages on the server
designed to download messages to the client
Poor handling of multiple client access
while many (most?) users have now multiple devices

Unlike POP, Internet Message Access Protocol (IMAP) was designed with multiple clients in mind

Support multiple mailboxes and searches on the server
client can create, rename, move mailboxes & search on server
Access to individual MIME parts and partial fetch
client can download only the text content of an e-mail
Support multiple clients connected to one mailbox
server keep state about each message (e.g. read, replied to)