Last Monday on Communication Networks

The WWW is made of three key components

- **Implementation**:
  - Clients/Browser
  - Servers
  - Proxies

- **Content**: Objects (files, pictures, videos, ...) organized in Web sites
  - Web sites (a collection of objects)

- **Infrastructure**:
  - URL: name content
  - HTTP: transport content

We’ll focus on its implementation

A Uniform Resource Locator (URL) refers to an Internet resource

protocol://hostname[::port]/directory_path/resource

Materials inspired from Scott Shenker and Jennifer Rexford

nsg.ee.ethz.ch
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

HTTP clients make request to the server

HTTP servers answers to clients’ requests

HTTP makes the client maintain the state.
This is what the so-called cookies are for!

<table>
<thead>
<tr>
<th>3 digit response code</th>
<th>reason phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1XX</td>
<td>informational</td>
</tr>
<tr>
<td>2XX</td>
<td>success</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>3XX</td>
<td>redirection</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>303</td>
<td>Moved Temporarily</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
</tr>
<tr>
<td>4XX</td>
<td>client error</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>5XX</td>
<td>server error</td>
</tr>
<tr>
<td>505</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>
Performance goals vary depending on who you ask

User
- wish
  - fast downloads
  - high availability

Network operators
- no overload
- cost-effective infrastructure

Content provider
- happy users
- infrastructure

Solution
- Caching and Replication
- Improve HTTP to compensate for TCP weak spots

Considering the time to retrieve $n$ small objects, pipelining wins

<table>
<thead>
<tr>
<th># RTTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-at-a-time</td>
</tr>
<tr>
<td>M concurrent</td>
</tr>
<tr>
<td>persistent</td>
</tr>
<tr>
<td>pipelined</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>$\approx 2n$</td>
</tr>
<tr>
<td>$\approx 2n/M$</td>
</tr>
<tr>
<td>$\approx n+1$</td>
</tr>
<tr>
<td>$2$</td>
</tr>
</tbody>
</table>

Considering the time to retrieve $n$ big objects, there is no clear winners as bandwidth matters more

<table>
<thead>
<tr>
<th># RTTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\approx n \times \text{avg. file size}$</td>
</tr>
<tr>
<td>bandwidth</td>
</tr>
</tbody>
</table>

To limit staleness of cached objects, HTTP enables a client to validate cached objects

Server hints when an object expires (kind of TTL) as well as the last modified date of an object

Client conditionally requests a resource using the "if-modified-since" header in the HTTP request

Server compares this against "last modified" time of the resource and returns:
- Not Modified if the resource has not changed
- OK with the latest version

Caching can and is performed at different locations

<table>
<thead>
<tr>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
</tr>
<tr>
<td>close to the client</td>
</tr>
<tr>
<td>close to the destination</td>
</tr>
</tbody>
</table>

- browser cache
- forward proxy
- Content Distribution Network (CDN)
- reverse proxy

Web

Video Streaming

HTTP-based

We want the highest video quality

Without seeing this …
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

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”
Buffer-based adaptation

[A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service, Huang et al., ACM SIGCOMM 2014]

Today 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

An e-mail is composed of two parts

E-mail

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
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 microsoft
  - multipart with subtypes mixed or alternative

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

Hi Tobias,

Here are some interesting questions…

Best,
Laurent

Common multipurpose Internet Mail Extensions

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

- **multipart/mixed**: multiple independent parts
  - e.g. plain text and a binary file
- **multipart/alternative**: multiple representation of the same content
  - e.g. plain text and HTML

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

**Content-Type** contains a parameter that specifies a string delimiter (usually chosen randomly by the client)

**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

<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>0x00</td>
<td>13</td>
<td>0x0d</td>
<td>31</td>
<td>0x1f</td>
<td>127</td>
<td>0x7f</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>14</td>
<td>0x0e</td>
<td>32</td>
<td>0x10</td>
<td>128</td>
<td>0x80</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>15</td>
<td>0x0f</td>
<td>33</td>
<td>0x11</td>
<td>129</td>
<td>0x81</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>16</td>
<td>0x10</td>
<td>34</td>
<td>0x12</td>
<td>130</td>
<td>0x82</td>
</tr>
</tbody>
</table>

If the length of the input is not a multiple of three, Base64 uses "=" as padding character.

**Binary input**: 0x14fb9c03d97e

<table>
<thead>
<tr>
<th>8-bits</th>
<th>6-bits</th>
<th>Decimal</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010100 11111011 10011100</td>
<td>00000011 11011001 01111110</td>
<td>5 15 46 28 0 61 37 62</td>
<td>F P u c A 9 1 +</td>
</tr>
</tbody>
</table>

<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>0x00</td>
<td>15</td>
<td>0x0f</td>
<td>31</td>
<td>0x1f</td>
<td>127</td>
<td>0x7f</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>16</td>
<td>0x10</td>
<td>32</td>
<td>0x10</td>
<td>128</td>
<td>0x80</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>17</td>
<td>0x11</td>
<td>33</td>
<td>0x11</td>
<td>129</td>
<td>0x81</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>18</td>
<td>0x12</td>
<td>34</td>
<td>0x12</td>
<td>130</td>
<td>0x82</td>
</tr>
</tbody>
</table>

If the length of the input is not a multiple of three, Base64 uses "=" as padding character.

**Binary input**: 0x14

<table>
<thead>
<tr>
<th>8-bits</th>
<th>6-bits</th>
<th>Decimal</th>
<th>base64</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010100</td>
<td>0000101</td>
<td>5 0</td>
<td>F A = =</td>
</tr>
</tbody>
</table>
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

MSA, MTA, MDA and MRA, MUA are often packaged together leading to simpler workflows

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)

<table>
<thead>
<tr>
<th>SMTP 3 digit response code</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2XX success</td>
<td>Service ready</td>
</tr>
<tr>
<td>250</td>
<td>Requested mail action completed</td>
</tr>
<tr>
<td>3XX input needed</td>
<td>Start mail input</td>
</tr>
<tr>
<td>354</td>
<td>Mailbox unavailable</td>
</tr>
<tr>
<td>4XX transient error</td>
<td>Insufficient space</td>
</tr>
<tr>
<td>421</td>
<td>Service not available</td>
</tr>
<tr>
<td>450</td>
<td>Insufficient space</td>
</tr>
<tr>
<td>452</td>
<td>Insufficient space</td>
</tr>
<tr>
<td>5XX permanent error</td>
<td>Syntax error</td>
</tr>
<tr>
<td>500</td>
<td>Unknown command</td>
</tr>
<tr>
<td>502</td>
<td>Bad sequence</td>
</tr>
</tbody>
</table>

server -- 220 hamburger.edu
EHLO crepes.fr
250 Mails crepes.fr, pleased to meet you
MAIL FROM: aclice@crepes.fr
250 Alice@crepes.fr... Sender ok
RCPT TO: bob@hamburger.edu
250 bob@hamburger.edu... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Do you like ketchup?
How about pickles?
.
250 Message accepted for delivery
QUIT
221 hamburger.edu closing connection
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

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

Try it out yourself!

```
telnet server_name 25
openssl s_client -starttls smtp
  -connect mail.ethz.ch:587
  -crlf
  -ign_eof
 perl -MMIME::Base64 -e 'print encode_base64("username");'
 perl -MMIME::Base64 -e 'print encode_base64("password");'
```
As with most of the key Internet protocols, security is an afterthought.

SMTP Headers
MAIL FROM: no checks are done to verify that the sending MTA is authorized to send e-mails on behalf of that address

Email content (DATA)
From: no checks are done to verify that the sending system is authorized to send e-mail on behalf of that address
Reply-to: ditto

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

Let's spoof some e-mails!

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

client
list
1 498
2 912
retr 1
<message 1 contents>
dele 1
retr 2
<message 2 contents>
dele 2
quit
OK POP3 server signing off

Authorization phase
Clients declares username and password
Server answers +OK/-ERR

+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 2 contents>
dele 2
quit
+OK POP3 server signing off

POP: Post Office Protocol
IMAP: Internet Message Access Protocol

Content | Infrastructure/Transmission | Retrieval
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)