Why HTTP needed extensions?

A taste of HTTP v1.1 additions

HTTP v1.1:
- introduces many complexities
- no longer an easy protocol to implement
Simple (one packet) Web Page Retrieval Time

- **SYN**
- **ACK+GET**
- **ACK+FIN**

Retrieval Time ≈ 2RTT + PTT
Bigger (7 packets) Web Page Retrieval Time

TCP Data Transmission Time $> 7$ Packet Transmission Time

Retrieval Time $\approx (2\text{RTT}) + (\text{TCP Data Trasm. Time})$
Complex (many objects) Web Page Retrieval Time with HTTP/1.0

\[\text{Time Retrieval} \approx 2n \text{ RTT} + \sum \text{TT}_i\]
Performance drawbacks

- **Mandatory roundtrips**
  - TCP three-way handshake
  - get request, data return
  - new connections for each image (parallelize)
  - lots of extra syn or syn/ack packets

- **Slow-start penalties**
  - Significantly affects fast networks

- **Lots of TCP connections to server**
  - spatial/processing overhead in server (TCP stack)
  - unfairness because of loss of congestion control info
Shorten Page Retrieval with HTTP/1.0

→ After the retrieval of the html source, start many parallel TCP connections to download the objects embedded

Time retrieval = 4 RTT + TT₁ + max(TTᵢ)

→ Overhead to manage n connections...

→ and slow starts significantly affect performance
Persistent HTTP

One TCP connection
Performance improvements

➤ Persistent HTTP
   ➤ in HTTP/1.0, add “Connection: Keep-Alive\r\n” header
   ➤ in HTTP/1.1, P-HTTP built in

➤ Does it help?
   ➤ server-side efficiency
      ➤ TCP memory consumption at server is the bottleneck
      ➤ … as well known by Denial Of Service attackers 😊
   ➤ allows multiple requests on one connection
      ➤ Faster retrieval

➤ Pipelining

G.Bianchi, G.Neglia, V.Mancuso
Pipelining

Without pipelining:
- GET html
- GET figure1
- GET figure2
- ...

With pipelining:
- GET html
- GET figure1
- GET figure2
- ...

G. Bianchi, G. Neglia, V. Mancuso
More complex handshake

- **Persistent HTTP**
  - Move task of closing TCP connection from Server to Client

- **how does a client know when document is returned?**
  - **Content-Length: header field:**
    - In HTTP v1.0 used only in POST requests
      - When client needs to send a body entity
    - In HTTP v1.1 must be used also in response
      - To inform the client that retrieval is finished

- **when does the connection get dropped?**
  - idle timeouts on server side
  - client drops connections
  - server needs to reclaim resources
Chunking

HTTP v1.0
- connection dropped = lost data
- no chunking

HTTP v1.1
- Concept of “chunking”
- Range: bytes=300-304,601-993
  - useful for broken connection recovery (like FTP recovery)
- “chunked” transfer encoding
  - segmenting of documents
  - don’t have to calculate entire document length.
  - useful for dynamic query responses.
Multi-homing

⇒ Multi-homing:
  ⇝ 1 IP address with multiple DNS names
  ⇝ ESSENTIAL for commercial deployment
     → Small companies need a web presence
     → With proper naming (e.g. www.joemushrooms.it)
     → But don’t have ICT resources to administer on their own
        » Solution: web-hosting companies

⇒ The problem of multi-homing
  ⇝ Web-host company: www.bianchihosting.it → IP=200.100.100.1
  ⇝ Customer1: www.joemushrooms.it
  ⇝ Customer2: www.frankbeans.it
  ⇝ Try resolving URLs:
     → http://www.bianchihosting.it/ ➔ http://200.100.100.1:80/index.htm
     → http://www.joemushrooms.it/ ➔ http://200.100.100.1:80/index.htm
     → http://www.frankbeans.it/ ➔ http://200.100.100.1:80/index.htm
  ⇝ ALL EQUAL! No way to differentiate them
     → unless using different ports, but solution hardly appreciated by customers
Multi-homing solution

- **Host:** `<web server name>`:
  - Example:
    - Host: www.joemushrooms.it
  - Additional header command in the HTTP request
  - Mandatory sent by client
    - Initially a patch in HTTP v1.0
    - Fundamental feature in HTTP v1.1

- **Addressing:**
  - Need a further level of indirection
  - Process listening on port 80 shall not be a web server
  - but a process redirecting the HTTP request to the proper server, based on host: content
  - (virtual host)
HTTP operation with intermediaries

Three form of intermediaries:

⇒ Tunnel
  → intermediary program acting as a blind relay between two connections.
  → once active, a tunnel is not considered a party to the HTTP communication (does not understands HTTP)
  → e.g: firewall, Network Address Translator, etc

⇒ Proxy
  → intermediary program acting as both server & client (see later).

⇒ Gateway
  → server acting as intermediary for some other server.
  → unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.
  → e.g. portals
Proxy

An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.
Usage: client-side portal through firewalls; helper applications to handle requests not supported by user agent

**Transparent proxy**: does not modify request or response beyond authentication and identification

**Non Transparent proxy**: provides added services (media transformation, anonymity filtering, protocol reduction)

G.Bianchi, G.Neglia, V.Mancuso
Caching

Fundamental in reducing Internet traffic

Proxy and Gateway may cache. Tunnel cannot.
Why caching?

- reduce latency
- reduce data traffic
- distribute (reduce) requests

HTTP: hierarchical caching
Some problems with caching

- **Consistency**
  - cached pages might not be the most recent...

- **Security**
  - what happens if I infiltrate a cache?
  - servers/clients don’t even know this is happening
  - e.g.: AOL used to have a very stale cache, but has since moved to Inktomi

- **Ad clickthrough counts**
  - how does Yahoo know how many times you accessed their pages, or *more importantly*, their ads?
Additional http v1.1 features
(mainly for caching purposes)

- **HTTP/1.0**: lots of problem associated to caching
  - Very poor cache consistency models
- **HTTP/1.1**: very improved caching model
  - Refer to RFC 2616 for details (too long to deal with here)

- **Additional header commands:**
  - **Age**: `<seconds, date>`
    - sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server
  - **Etag**: fa898a3e3
    - unique tag to identify document (strong or weak forms)
  - **Cache-control**: `<command>`
    - marking documents as private (don’t keep in caches)
Caching saving

some numerical examples
Example 1

Server Response Time ($ST_1$) = 20ms
Assume 3*1460B pages
1500B packet size, 40B header
Consider only bottleneck transmission time
Web Page Retrieval

SYN
ACK+GET
ACK
ACK

SYN+ACK

3 RTT + 3 PT + ST

G.Bianchi, G.Neglia, V.Mancuso
LAN (100Mbps) → 1Mbps → Public Internet

RTT_1 (without congestion) = 100 ms

PT_1 = 12 ms, ST_1 = 20 ms

Retrieval Time (1) = 3*100 + 3*12 + 20 = 356 ms
Add a cache

- LAN 100Mbps
- Public Internet 1Mbps
- Proxy/cache

RTT₂ = 5ms
Hit rate = 20%
ST₂ = 20ms  PT₂ ≈ 0ms

Retrieval Time(2) = 3*5 + 3*0 + 20 = 35ms
LAN 100Mbps

Proxy/cache

Public Internet

1Mbps

RTT \_2 = 5\text{ms}

Hit rate = 20% 

Average Retrieval Time = 0.2*35+0.8*356= 292 ms

64 ms saving! (18%)
More insight

Server Time ($ST_1$) = 20ms

Assume 3*1500B pages
Web Page Retrieval

SYN
QD
ACK+GET
QD
ACK
QD
ACK
ACK

3 RTT + 3 QD + 3 PT + ST

G.Bianchi, G.Neglia, V.Mancuso
Example

LAN (100Mbps) 1Mbps

RTT₁ (without congestion) = 100ms

PT₁ = 12 ms

QD₁ = ?
Bits of Queueing Theory
M/M/1/$\infty$

- interarrival time and packet size exponentially distributed

$R$ = average rate
$C$ = link capacity

G.Bianchi, G.Neglia, V.Mancuso
Bits of Queueing Theory

$M/M/1/\infty$

$R/C = \text{offered load, an important quantity in queueing theory}$

$$N = \frac{R/C}{1 - R/C} \quad \text{Average packet \# in the queue}$$

$$QD = \frac{N}{C} \quad \text{Average queueing delay (if C packet/s)}$$

$$QD + \frac{1}{C} = \frac{1/C}{1 - R/C} \quad \text{Average service time}$$
Bits of Queueing Theory
M/M/1/∞
Example

LAN (100Mbps) 1Mbps Public Internet

RTT (without congestion) = 100ms

Congestion Delay

QD₁ = ?

• ≈ 36kbit/request
• 26 request/s

936kbps → R/C = 93.6% → QD₁ = 176ms

Retrieval Time = 3*100 + 3*176 + 3*12 + 20 = 884ms > 356ms
Example

- LAN 100Mbps
- Public Internet 1Mbps
- Proxy/cache

Hit rate = 20%
RTT₂ = 5ms
ST₂ = 20ms
PT₂ ≈ 0ms
QD₂ ≈ 0ms

Retrieval Time(2) = 3*5 + 3*0 + 3*0 + 20 = 35ms
Example

- 36kbit/request
- 21 request/s

QD \_1 = ?

Retrieval Time(1) = 3*100+3*34+3*12+20=458ms < 884ms

\[ \rho = 74\% \implies QD_1 = 34ms \]

PT_1 = 12 ms \quad ST_1 = 20 ms \quad QD_1 = ?

RTT_2 = 5ms

LAN 100Mbps

Public Internet

Proxy/cache

1Mbps

G.Bianchi, G.Neglia, V.Mancuso
Example

Hit rate = 20%
Retrieval Time(1) = 35ms  Retrieval Time(1) = 456ms

Average Retrieval Time = 0.2*35+0.8*456 = 372ms

512 ms saving! (58%)
Some HTTP examples

- Download an image
- Need for Host in http/1.1
- Connection: permanent/close
- HTTP stateless
- Conditional GET