HTTP/2.0: Intermediary Requirements

Willy Tarreau - Exceliance (HAProxy project)

Amos Jeffries - Treehouse Networks Ltd. (Squid project)

*Hit the space bar for next slide*
Issues intermediaries are currently facing

Intermediaries have a complex role:

- must support unusual but compliant message formatting (eg: variable case in header names, single LF, line folding, variable number of spaces between colon and field value)
- fix what ought to be fixed before forwarding (eg: multiple content-length and folding), adapt a few headers (eg: Connection)
- must not affect end-to-end behaviour even if applications rely on improper assumptions (effects of rechunking or multiplexing)
- need to maintain per-connection context as small as possible in order to support very large amounts of concurrent connections
- need to maintain per-request processing time as short as possible in order to support very high request rates
- front line during DDoS, need to take decisions very quickly
A few numbers taken from haproxy experience reports give a better idea of the world of intermediaries:

- 750 CPU cycles: the time it takes to parse an average HTTP request
- 6 microseconds: total CPU time (usr+sys) spent on forwarding a complete end-to-end request+response from `accept()` to `close()` (~18k CPU cycles @3 GHz)
- 30 sec: the most common request/response timeout setting
- 17 kB: the most common per-request context, out of which 16 kB are used to store headers to be parsed
- 1 million: the largest reported number of peak concurrent connections on a single server (long polling)
- 10 million/sec: the largest reported peak connection rate during a 24 Gbps DDoS, 60 machines involved to stand the load.
What intermediaries need

Intermediaries would benefit from:

- Reduced connection/requests ratio (more requests per connection)
  → drop of connection rate
  → drop of memory footprint (by limiting concurrent conns)

- Reduced per-request processing cost and factorize it per-connection
  → higher average request rate
  → connection setup cost is already "high" anyway

- Reduced network packet rate by use of pipelining/multiplexing
  → reduces infrastructure costs
  → significantly reduces RTT impacts on the client side
Step 1 : reduce message parsing costs

Several changes must be applied to reduce message parsing costs :

• no more parsing to skip over useless information - all message elements must have a known length (known by design or advertised).

• no more parsing/lookup of usual methods and header names, use enums for most common ones.

• no more painful parsing of chunk size + CRLF + extensions, just binary-encode the length
Step 2: factor out redundant processing

Most consecutive requests from a user agent share everything but the URI. Factor them out by introducing two new header sections:

- per connection: User-Agent, Host, etc... are commonly unchanged over a same connection. → transport-section
- per group of messages: Accept, Referer, Cookies, are commonly unchanged for a number of consecutive requests. → common-section
- the rest is the message section

→ significant reduction of upstream bandwidth
Step 3: encourage out-of-order processing and multiplexed connections

Pipelining and out-of-order processing reduce the number of round trips required to transmit all the requests and responses.

- significant reduction of round trips.
- more full network packets, less total packets.
- 16-bit request ID allows up to 64k outstanding requests per connection.
- even with HTTP Upgrade+101 there are solutions to save round trips
Example : before reduction

Example on IETF83 page : 18 requests similar to this one totalize 7582 bytes upstream to load the whole page :

```
GET /css/ietf.js HTTP/1.1
Host: www.ietf.org
User-Agent: Mozilla/5.0 (X11; Linux i686 on x86_64; rv:6.0.2) Gecko/20100101 Firefox/6.0.2
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://www.ietf.org/meeting/83/
Cookie: styleSheet=1
```
Example: after reduction

Once reduced using the two methods above, the 18 requests become 707 bytes (91% compression ratio):

- **172 + 1 bytes of transport section**:
  - 1 byte for User-Agent, 1 byte for length, 78 bytes for value
  - Same for Host, Connection, Accept-Language, Accept-Encoding and Accept-Charset (total 92 bytes)

- **47 + 1 bytes of common section (2nd and further requests)**:
  - 1 byte for Cookie, 1 byte for length, 12 bytes for value
  - 1 byte for Referer, 1 byte for length, 31 bytes for value

- **468 + 18 bytes of message sections**:
  - 18 * 1 byte for GET + HTTP version
  - 18 * 1 byte for URI length
  - 432 bytes for all 18 URIs
Benefits

• Cheap to emit for clients (UA and intermediaries), only factor out what is certain
• Mobile-networks friendly by large reduction of volume (18 requests fit in half an MSS)
• Cheap to parse for intermediaries, as already known headers don't have to be parsed again
• Cheap to store for intermediaries, as it helps reducing the per-request context
• Lower latency out-of-order delivery using a simple request ID
• Low forwarding cost with easy memcpy()/writev() from transport/common section to the message
Next steps

- Finer impact estimates on intermediaries / end points
- More work needed on the handshake
- Finish the draft with all the details
- Get more intermediaries and client authors involved
- Have a working implementation for tests and measurements