CSE 124: HTTP AND THE WEB

George Porter
Oct 4, 2017
ATTRIBUTION

- These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
- These slides incorporate material from:
  - Computer Networks: A Systems Approach, 5e, by Peterson and Davie
Outline

1. Finishing up framing and parsing
2. HTTP protocol
3. Demo: interacting with web servers
DEFINITIONS

• **Operation (e.g., in a voting system)**
  • An action you can perform within a protocol’s peer interface
  • E.g., “Submit vote”, “get current vote count”, “reset vote count to zero”

• **Message**
  • An encoding of an operation according to a protocol’s *wire format*. Common formats include XML, binary, JSON, ...

• **Framing**
  • Writing out (and reading in) messages from a stream such that messages can be separated and interpreted correctly

• **Parsing/encoding/decoding**
  • Converting a message to/from an application data structure
EN/DECODING APP STATE TO/FROM A MESSAGE

- Binary
- Text (ad-hoc)
- Text (XML)
- Many others...

```c
struct Employee_t {
    uint8_t operation;
    uint64_t id;
    uint16_t department;
};
typedef struct Employee_t Employee;

// list of operations
enum {
    ADD_EMPLOYEE = 1,
    QUERY = 2,
    DELETE_EMPLOYEE = 3;
};
```

Operation: 1, id=428, d=80

```
<employee>
    <operation>1</operation>
    <id>428</id>
    <department>80</department>
</employee>
```
FRAMING: LENGTH SPECIFICATION VS DELIMITERS

- Binary representation of name?
  - Handling variable length
- Consider "Alan" as a name

```
struct Employee_t {
    uint8_t operation;
    char * name;
    uint64_t id;
    uint16_t department;
};
typedef struct Employee_t Employee;
```

- Option 1: Explicit length
  - But how big should length be?

```
enum {
    ADD_EMPLOYEE = 1,
    QUERY = 2,
    DELETE_EMPLOYEE = 3;
};
```

- Option 2: Delimiter
  - But what if delimiter is in the message?
FRAMING: GETNEXTMSG AND PUTMSG

• GetNextMsg()
  • Finds and returns bytes corresponding to single message
  • Even if messages are variable length

• PutMsg()
  • Writes out bytes corresponding to a message with enough context for GetNextMsg to work
FRAMING: SUMMARY

- **PutMsg()**
  - Given an array of bytes representing an application-level operation, writes to stream
    1. **Explicit length**
      - Writes out the length of the message, then message
    2. **Delimiter**
      - Ensures delimiter doesn’t appear in message
      - Writes out message
      - Then writes out delimiter

- **GetNextMsg()**
  - Reads from stream until entire message is read, returns to higher layer
    1. **Explicit length**
      - Reads the length, then reads that many bytes (security?)
    2. **Delimiter**
      - Reads continuously into a buffer until delimiter is encountered
      - Message then returned to higher layer
FRAMING DEMO

• Code at https://github.com/gmporter/cse124-lec-protocols
Outline

1. Finishing up framing and parsing
2. HTTP protocol
3. Demo: interacting with web servers
HTTP AS AN EMERGING TRANSPORT LAYER

- HTTP: HyperText Transfer Protocol
  - Tim Berners-Lee at CERN in 1989

- In addition to web browsing:
  - Video streaming via DASH on YouTube.com
  - REST (Representational state transfer)
  - Chat apps like Slack
  - Many others
• Documents link to other documents
  • Specified in HTML files
• HTTP is the protocol for retrieving HTML files from servers
  • and images, sounds, video, ...
• Implemented in servers
  • Apache, nginx, MSFT IIS
• and clients
  • Chrome
  • MSFT Edge
  • Apple Safari...
HTTP OVERVIEW

- HTTP is a text oriented protocol.
- HTTP is a request/response protocol.
- Requests and responses both look like:

  START_LINE <CRLF>
  MESSAGE_HEADER <CRLF>
  <CRLF>
  MESSAGE_BODY <CRLF>

- The first line (START LINE) indicates whether this is a request message or a response message.
HTTP REQUESTS

• Request Messages define
  • The operation (called *method*) to be performed
  • The web page the operation should be performed on
  • The version of HTTP being used.
• Examples:
  • GET /index.html HTTP/1.0
  • GET /images/catimg23.jpg HTTP/1.1
  • GET /contracts/contract3.txt HTTP/1.1
OPTIONAL HTTP REQUEST HEADERS

• After the start line are *request headers*:
  • Text-based, key and value separated by a colon

• Example 1:

  GET /index.html HTTP/1.0
  User-Agent: Firefox 23.3.1

• Example 2:

  GET /images/cat2.jpg HTTP/1.1
  Host: www.cs.ucsd.edu
  User-Agent: Chrome 12.1
HTTP RESPONSES

- Also begins with a single START LINE.
  - The version of HTTP being used
  - A three-digit status code
  - Text string giving the reason for the response.

- Example:
  
  HTTP/1.1 200 OK
  Content-Type: text/html
  Content-Length: 291
It works!

Connection closed by foreign host.
borabora:~ gmporter$
## HTTP RESPONSE CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Example Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Informational</td>
<td>request received, continuing process</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>action successfully received, understood, and accepted</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>further action must be taken to complete the request</td>
</tr>
<tr>
<td>4xx</td>
<td>Client Error</td>
<td>request contains bad syntax or cannot be fulfilled</td>
</tr>
<tr>
<td>5xx</td>
<td>Server Error</td>
<td>server failed to fulfill an apparently valid request</td>
</tr>
</tbody>
</table>

- For project 1:
  - 200: OK
  - 400: Client Error
  - 403: Forbidden
  - 404: Not Found
HTTP PIPELINING (VERSION HTTP/1.1)

- HTTP/1.0 opened a new connection for every data item it retrieved
- Overhead in establishing a new connection to the same server over and over again
- HTTP/1.1 Persistent Connections
  - Reuse connection over many requests/responses
  - But more complex in terms of framing/parsing
    - How to know when one request ends and the next begins?
    - This is part of the 1.1 spec
REQUIRED REQUEST HEADERS (AT LEAST FOR 124)

• **Host:**
  • Indicates the name of the server you are accessing
  • Used to implement virtual hosts

• **User-Agent:**
  • Identifies what software is issuing the request
  • E.g.:
    • User-Agent: Opera/9.25 (Windows NT 6.0; U; en)
    • User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/125.2 (KHTML, like Gecko) Safari/125.8
REQUIRED RESPONSE HEADERS (AT LEAST FOR 124)

- **Server:**
  - Identifies the server
    - Server: Apache/2

- **Content-Length:**
  - How many octets (byte) in the response

- **Content-Type:**
  - text/html
  - image/jpeg
  - image/png
Outline

1. Finishing up framing and parsing
2. HTTP protocol
3. Demo: interacting with web servers
Usage:

```
curl -v -o /dev/null http://<URL>
```