Protocols of IoT: HTTP

e-Yantra Team

Embedded Real-Time Systems (ERTS) Lab
Indian Institute of Technology, Bombay

IIT Bombay
July 5, 2022

Agenda for Discussion

1. Intro to HTTP
2. HTTP Overview
3. HTTP in Detail

What is a protocol and why do we need one?

What is a protocol and why do we need one?

To effectively communicate, the conversing entities need to know a common language.

Without it, effective exchange of information won't happen.

A protocol is that common language or set of rules that the entities need to know.

Hypertext Transfer Protocol (HTTP)

HTTP, stands for Hypertext Transfer protocol.

Protocol of the web.

Computers on the web can send data to each other using HTTP.

Hypertext Markup Language (HTML)

Hypertext Markup Language (HTML)

```
<!doctype html>
<html>
    <head>
        <title> Susan's Website </title>
        <link rel="stylesheet" type="text/css" href="example.css">
    </head>
    <body>
        <div class="header">
            Susan Baugh
        </div>
        <div class="content">
            <h1>Welcome!</h1>
            <p>
                Thanks for stopping by!  My name is Susan and I love
                    art, learning about energy and renewable
                    resources, and playing Candy Crush. <br>
                Please have a look around and check out my projects
                    and work! <br>
            </p>
            <img src="http://i.imgur.com/sOgMR3u.jpg" alt="I'm Susan"
                width=200/>

            <h1>Publications</h1>

            <h2>Presentations</h2>

            Construction of the Belo Monte Hydroelectric Power Plant
                in the Amazon <br>
```

Figure 1: Source HTML

Susan Baugh

Welcome!

Thanks for stopping by! My name is Susan and I love art, learning about energy and renewable resources, and playing Candy Crush.
Please have a look around and check out my projects and work!

Publications

Presentations

Construction of the Belo Monte Hydroelectric Power Plant in the Amazon
Microfluidic Devices for Medical and Energy Applications

Figure 2: Rendered Output

Hypertext Transfer Protocol (HTTP)

Used to transfer Hypertext documents (typically HTML pages), that can contain text, images, audio, video, styles, scripts, or links (hyperlinks) to other hypertext documents.

HTTP is a generic, stateless protocol.

It is an application layer protocol.

HTTP is a request-response client-server protocol.

Why are we learning HTTP?

1. Most important protocol of the Internet.
2. A lot of external services on the web provide an HTTP API.
3. To publish telemetry (sensor) data and listen for commands from web servers.
4. IoT devices can leverage existing services by using HTTP.

Key Terms

1. Client
2. Server
3. Request
4. Response
5. Resource
6. Message
7. Connection

Client and Server

Two important entities: **Client** and **Server**

A web server is a provider of services while a client is a requestor of the services.

Client and server both are processes running on computers. A computer can host (run) many clients and servers.

HTTP server implicitly runs on port 80.

Request and Response

HTTP is a request-response client-server protocol.

Client sends a request for data (HTML, CSS, JS, JSON, Binary files). A web server services the request and sends data in response.

HTTP is a pull protocol, the client *pulls information from the server (instead of server pushing information down to the client).

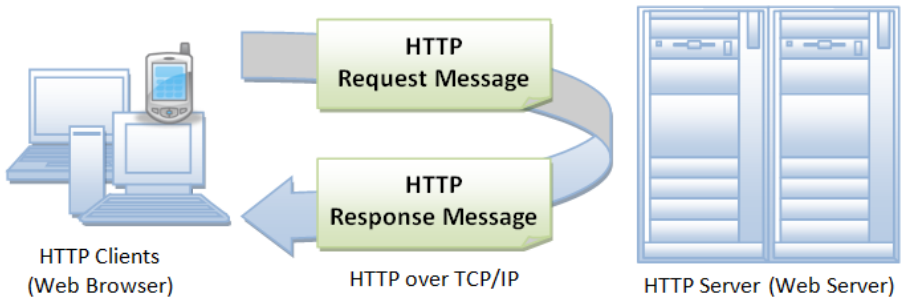

Figure 3: Client-Server and Request -Response

Resource

The web contains lots of servers. A server can contain lots of information. How does a client *request a specific piece of information in that case?

It does so by sending a request on a particular **resource**. An HTTP request is always **targetted to a resource**.

It is identified by a **Uniform Resource Locator (URL)**, known as the web address. Format: protocol://hostname:port/path-and-file-name

E.g.: `http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument`

Parts of a resource locator

scheme://authority:port/path?query#fragment

1. Scheme (http or https)
2. Authority (www.google.com)
3. Port (80, 8080, 3001, etc.)
4. Path
5. Query (key1=value1&key2=value2)
6. Fragment

Resource

Examples

https://www.google.com/search?sxsrf=ACYBGNQAZ9ZC_m8Sg5kohN3z4WTYUI1WGg%3A1580104209880&source=hp&ei=EXouXsvdM7-e4-EPpeSGiAY&q=hello+world&oq=hello+world&gs_l=psy-ab.3..0l10.726.3433..3579...0.0..0.301.1861.5j5j2j1.....0....1..gws-wiz.....35i39j0i131j0i67j0i20i263.1IvM-ut4B1Y&ved=0ahUKEwiLqovxiqPnAhU_zzgGHSWyAWEQ4dUDCAU&uact=5

https://www.e-yantra.org/#eLSI

http://eyic.e-yantra.org/#section2

Multiple types of Operations

A news service may let a user (a journalist) add, delete or update the information he/she owns on the server. A social media service lets a user publish, edit and delete his/her status updates.

HTTP provides support for these different operations via HTTP Request Methods (also called verbs).

There are methods for **GET**ting, **POST**ing, **PATCH**ing, **DELETE**ing information.

HTTP GET

HTTP GET method requests data at the specified resource.

Requests using GET should only retrieve data.

Syntax: GET /index.html HTTP/1.1

Example applications: Opening up a web page, downloading a file, mobile app fetches data from web

HTTP POST

HTTP POST method creates new data on the server. Type of data is indicated by Content-Type header. Commonly through an HTML Form.

Syntax: POST /test HTTP/1.1

Common content types:

- application/x-www-form-urlencoded
- application/json
- multipart/form-data
- text/plain
- application/octet-stream

Example applications: Login, Uploading a file, Creating a post on social media.

HTTP POST Content-Types

1. **application/x-www-form-urlencoded**

   field1=value1&field2=value2

2. **application/json**

   {"field1": "value1", "field2": value2}

HTTP PUT

HTTP PUT method creates a new resource or replaces a representation of the target resource with the request payload.

Calling PUT once or several times successively has the same effect (that is no side effect), BUT successive identical POST may have additional effects.

Syntax: PUT /new.html HTTP/1.1

Example applications: Update user details

HTTP DELETE

HTTP DELETE method deletes the specified resource.

Syntax: DELETE /file.html HTTP/1.1

Example applications: Delete entities on the web server.

Specifying Options and Configuration

Today, a lot of devices like mobiles, tablets, fitness watches, smart TVs, IoT devices, laptops and computers access the web.

All devices might not support all **media types** (cool way of saying file types) depending on their platform.

Many times, the servers and clients are also present in different countries. A client should be able to specify the language it understands.

HTTP is flexible, configurable and extensible through use of **Headers**.

Examples of headers: Accept and Accept-*, Content-*, User-Agent, Location.

Reporting success or failure

A request can succeed or fail. How does a client know about it?

HTTP provides status codes on responses.

A status code indicates if the response succeeded or failed, the reason for failure (client not authorised, error in request, server issue).

HTTP in Detail

HTTP Request Message Format

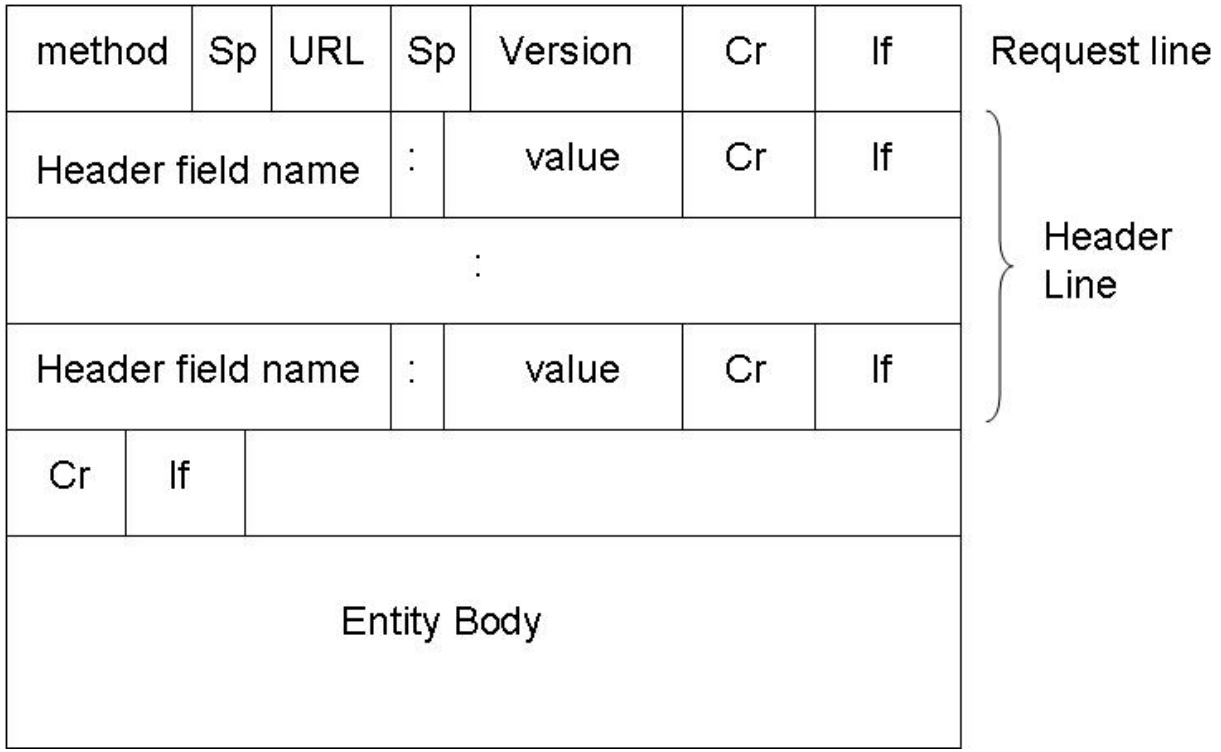

Figure 4: HTTP Request Message

HTTP Request Message Format

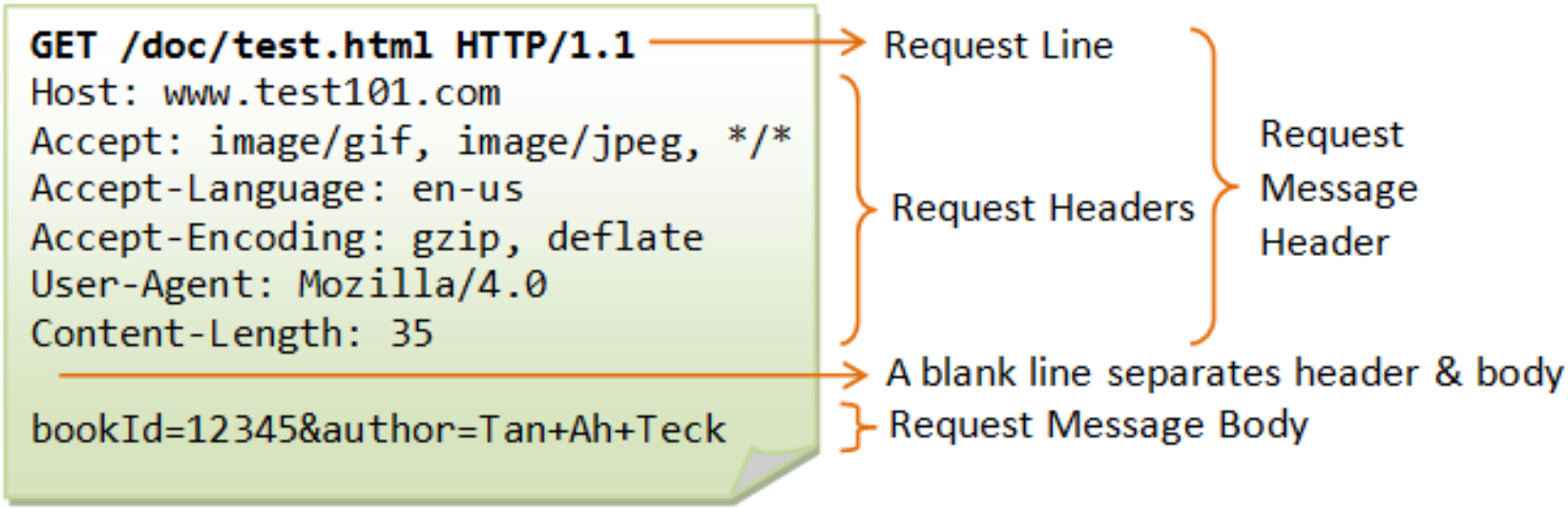

Figure 5: HTTP Request Message

HTTP Response Message Format

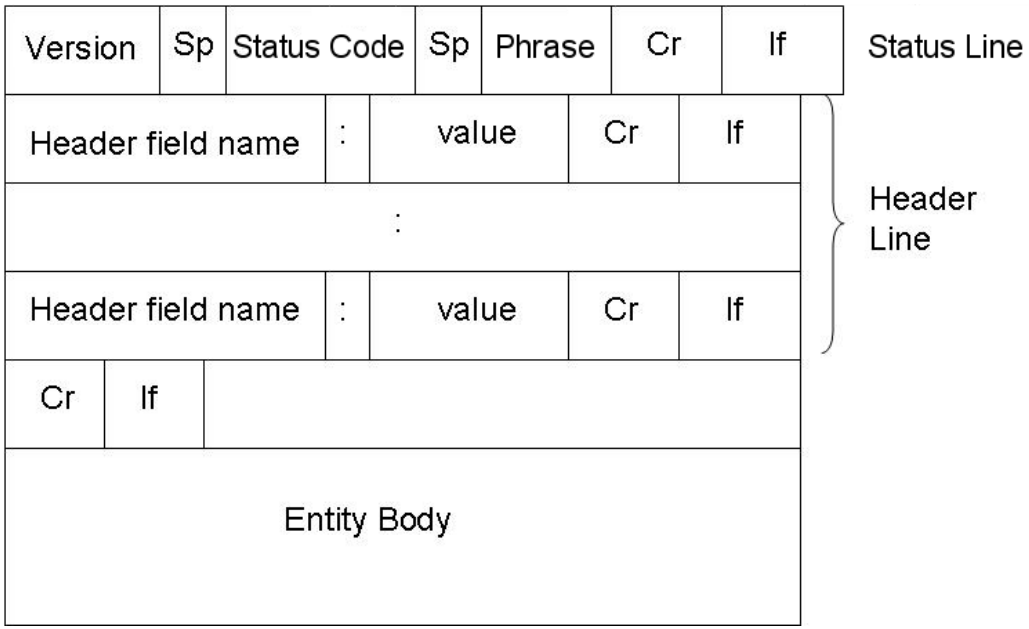

Figure 6: HTTP Response Message

HTTP Response Message Format

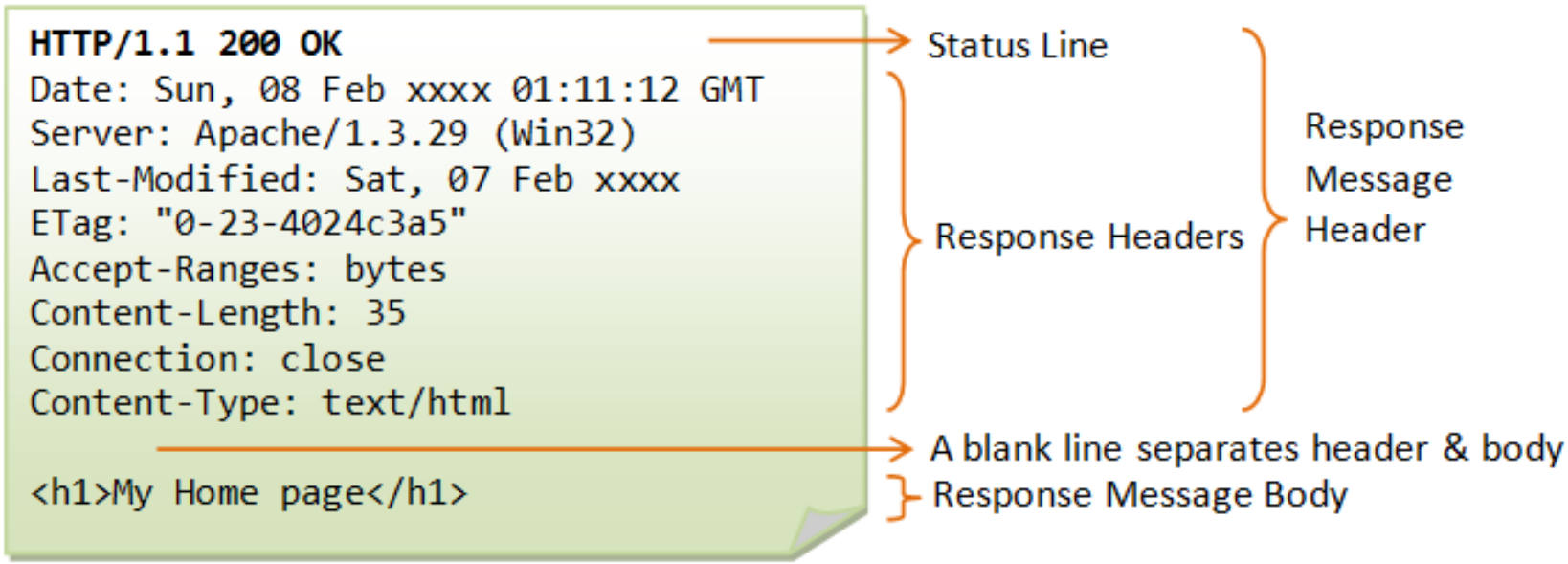

Figure 7: HTTP Response Message

Let's see HTTP messages

- Open terminal and type **curl -v www.google.com**

HTTP Response Status Codes

HTTP Response consists of a **status code, headers and the response body**.

Indicates whether the HTTP request completed successfully or not.

Responses are grouped in five classes:

1. Informational responses (100 - 199)
2. Successful responses (200 - 299)
3. Redirects (300 - 399)
4. Client errors (400 - 499)
5. Server errors (500 - 599)

HTTP Response Status Codes

Most recurring ones are:

1. Success Codes:
   1. 200 OK
   2. 201 Created
   3. 202 Accepted
2. Redirection Messages:
   1. 301 Moved Permanently
   2. 304 Not Modified
   3. 307 Temporary Redirect

HTTP Response Status Codes

Most recurring ones are:

3. Client Errors:
    1. 400 Bad Request
    2. 401 Unauthorized
    3. 403 Forbidden
    4. 404 Not Found
    5. 405 Method not Allowed
    6. 408 Request Timeout

HTTP Response Status Codes

Most recurring ones are:

4. Server Errors:
    1. 500 Internal Server Error
    2. 502 Bad Gateway
    3. 503 Service Unavailable
    4. 504 Gateway Timeout

HTTP Headers

Headers let the client and the server pass additional information with an HTTP request or response.

It's definition consists of it's case-insensitive name followed by a colon (:), then by its value.

Proprietary headers use "X-" prefix

Types of headers:

1. General
2. Request
3. Response
4. Entity

General Headers

A general header can be used in both request and response message but don't apply to the content itself.

1. Date
2. Connection

Request Headers

A request header can be used in an HTTP request, and doesn't relate to the content of the message.

1. Accept and Accept-* (Content negotiation)
2. Host
3. If-* (Caching)
4. Cookie
5. User-Agent

Response Headers

A response header can be used in response messages but doesn't apply to the content itself.

1. Age
2. Location
3. Server
4. Etag
5. Last-modified
6. Set-Cookie

Entity Headers

An entity header describes the content of the body of the message. Used in both requests and responses.

1. Content-Type
2. Content-Length
3. Content-Encoding

Request-Response Body

Not all requests and responses have a body. Requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one.

Most common request-response body types (specified using Content-Type header):

1. text/html
2. application/json
3. application/x-www-form-urlencoded
4. multipart/form-data

Request-Response Body

Not all requests and responses have a body. Requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one. Most common request-response body types (specified using Content-Type header):

5. image/png
6. text/css
7. application/javascript
8. application/octet-stream

e-Yantra IoT Framework

Ex. Weather Station

1. Database: **Google Sheets**
2. Alert Notifications: **Google Apps Scripting**
3. IoT Dashboard: **Grafana**

APIs

Application Programming Interface Using this interface we can program user applications that will use services residing on the web. Example:

1. Automatically create N github repositories using Github API
2. Push IoT sensor data to Google Sheets using Google Sheets API
3. Integrate data from weather API to make decisions for your IoT powered farm

Uses HTTP for transport (obviously).

APIs (Contd.

1. **WiFi**
   1. void **ey_init_wifi_sta**(char str_ssid[], char str_pass[])
   2. void **ey_wifi_auto_reconnect**(void)
2. **Database**
   1. int **ey_push_sensor_data**(unsigned char mode, float sensor_value);
   2. int **x_ey_push_sensor_data**(unsigned char mode, int number_of_values, ...);

References

1. MDN `https://developer.mozilla.org/en-US/`
2. NTU `https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html`
3. Insomnia `https://insomnia.rest/`
4. cURL `https://curl.haxx.se/`
5. Fake JSON Server `https://github.com/typicode/json-server`

Thank You!

Post your queries on: helpdesk@e-yantra.org