Web browers and servers

- A web browser is a program to retrieve and display resources on the Web
  Example: Netscape 1.0N

- Popular web browsers: Google Chrome, Apple Safari, Microsoft Internet Explorer, Mozilla Firefox
  - https://www.w3counter.com/globalstats.php
- A web server receives requests from a web browser and returns the requested resource.
  - Retrieve the resource from file system on server
  - Run a program to generate the resource
- Popular web servers: Microsoft IIS, Apache HTTP server, Nginx
  - https://news.netcraft.com/archives/2021/08/25/august-2021-web-server-survey.html

Web resources

- A resource is anything that is important enough to be referenced as a thing in itself.
- A resource has at least one URL as its address. A browser uses the URL to download the resource from a server.
- The web server uses MIME type to specify the data type of a representation of a resource.
  - Examples of resources:
    - The front page of SUSTech web site
    - The logo of SUSTech
    - A directory of resources pertaining to "web design" found by Google.
    - A json record returned by Facebook graph API: https://graph.facebook.com/cocacola

Address of web resource

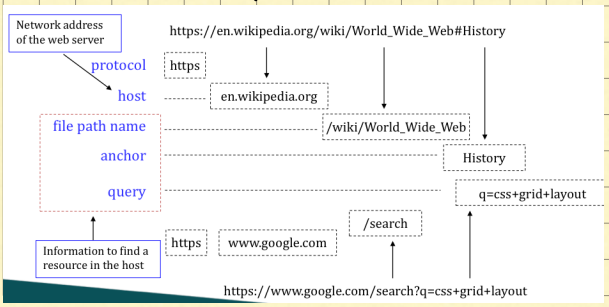
- Each web resource has a **URL** (Universal Resource Locator) as an address. It includes
  - protocol – how to communicate with the server
  - address of a web server – where to find the resource
  - additional info for the server to find the resource – which resource in the server
- Given a URL, a browser has enough info to construct a request to retrieve the resource

Syntax of URL

protocol://host/filepathname?query#anchor

- Protocol is either 'http' or 'https'
- Host is usually a domain name of a web server
- File path name identifies a resource in the server
- Query is form data submitted to a server-side script. The format is name1=value1&name2=value2
- Anchor refers to an element with the id in an html or xml file.
- Notice that separator characters in blue must be encoded if they are used as ordinary text in URL.

Inside a URL



Encode characters in URL

- In general, characters other than letters and numbers should be encoded in URL.
  - Space may be written as '+' or '%20'
  - Other printable character in ASCII should be written in hexadecimal. e.g, 1/2 is encoded as '1%2F2'
  - Non-ascii characters, e.g, Chinese, should be written as UTF-8 in hexadecimal. e.g.中文 is encoded as '%E4%B8%AD%E6%96%87'

Data type of web resource

- There are various types of web resources
  - E.g. text, image, audio, video, data
- There are different formats to encode a certain type
  - E.g. an image can be in GIF, JPEG or PNG
  - Some formats are defined by W3C, e.g. HTML, CSS, XML (general data), PNG (bitmap image), SVG (vector graphics)
  - Others are de-facto standards defined by the industry, e.g. GIF, JPEG, SWF (flash movie), JavaScript
- These formats are identified by a standard called **MIME type**
- Most browsers support these formats, and can display them correctly.

Common MIME types

category	MIME type	format
Text	text/html	HTML (.html)
	text/css	CSS (.css)
Images	image/gif	GIF (.gif)
	image/jpeg	JPEG (.jpg)
	image/png	PNG (.png) Portable Network Graphics
	image/svg+xml	SVG (.svg) Scalable Vector Graphics
Audio	audio/mp3, audio/m4a	Mp3 audio, MPEG4 audio
Video	video/mp4, video/m4v	MPEG4 video
Multipurpose	application/xml	General XML
	application/json	JSON data
	application/javascript	JavaScript source (.js)
	application/octet-stream	Arbitrary binary data
	application/x-www-form-urlencoded	HTML form submission

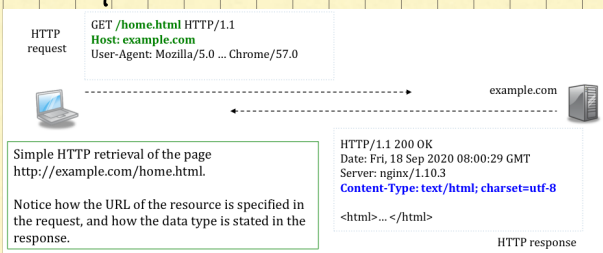
HTTP Operation

- Each HTTP transaction consists of one request and one response
  - A client sends a request for a resource to the server. Then the server returns a response containing a representation of the resource to the client.
- HTTP/1.0 requires a new TCP connection for each HTTP transaction
  - the server closes the connection after sending the response
- HTTP/1.1 can transfer several web resources in a TCP connection

Basic retrival

- To retrieve a static file in server
  - A browser sends a request to GET a resource at a URL
  - The server receives the request, maps the URL to a file in its file system, e.g. http://example.com/home.html -> c:\inetpub\home.html
  - The server infers the MIME-type from the file extension, e.g. *.html -> text/html, *.gif -> image/gif
  - The server constructs and returns the response

Example

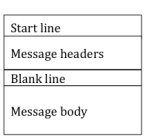


Basic HTTP request & response

- A basic **request** contains
  - Method GET to retrieve a resource
  - URL (or part of URL) – address of the resource
  - Other headers, e.g. name of user-agent
- A basic **response** contains
  - Status code
  - A representation of the resource
  - its MIME-type and encoding (for text resource) in Content-type header
  - Other headers, e.g. name of server

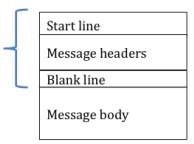
HTTP message structure

- **HTTP requests and responses have similar structure**
  - **Start line** – URL requested, any error
  - Headers – additional info
  - **Blank line**
  - Body – representation of the resource
- **Start line** and **blank line** are **required**.
- Headers and body are **optional**.



- Start line and headers must be in US-ASCII
  - i.e. Chinese characters must be encoded
- Message body can be text in any encoding or binary data
  - The header Content-Type defines the data type
  - The header Content-Length defines the size (in bytes)

Each line ends with CR LF (ASCII 13 and 10), and can only contain US-ASCII characters.



The body can be in any encoding or binary data

Request Line

- The request line has three parts:
  - Method name: GET, POST, HEAD, etc
  - (partial) URL of the resource
  - Version of HTTP: HTTP/1.0 or HTTP/1.1

```
GET /test.html HTTP/1.0
```

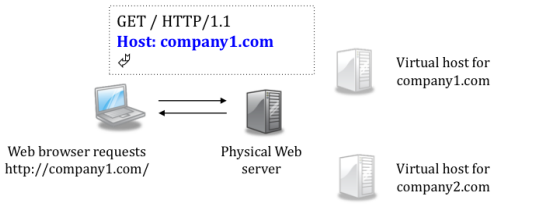
```
GET http://www.example.com/test.html HTTP/1.0
```

```
GET /search?hl=en&q=HTTP+headers&btnG=Google+Search HTTP/1.1
```

```
POST /accounts/ServiceLoginAuth?service=mail HTTP/1.1
```

Host

- Compulsory header in HTTP/1.1
- Specifies the host of the requested URL
  - Useful for multi-homed web servers. Several web sites (with different host names) may be using the same IP address.



Example

To retrieve the web page
http://www.example.com/test.html
The web browser constructs a HTTP request and sends it to the web server at www.example.com.

```
GET /test.html HTTP/1.1
Host: www.example.com

```

```
GET /test.html HTTP/1.0

```

```
GET http://www.example.com/test.html HTTP/1.1
Host: www.example.com

```

After receiving the request for
http://www.example.com/test.html
The web server will return a response containing the page, or an error message if it cannot find the page.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 2300

<html>... content of test.html ...</html>
```

```
HTTP/1.1 404 Not found
Content-Type: text/html
Content-Length: 1024

<html>... error message ...</html>
```

Status Line

- The status line has three parts:
  - HTTP version
  - Response status code
    - 200 – success
    - 404 – file not found
    - 500 – server error
  - English reason phrase, which describes the status code

```
HTTP/1.1 200 OK
```

```
HTTP/1.1 404 Not found
```

- HTTP defines five classes of status code
  - 1xx – informational message
  - 2xx - success of some kind — Here you go.
  - 3xx – redirects the client to another URL — Go away.
  - 4xx – an error on the client's part — you screwed up.
  - 5xx – an error on the server's part — I screwed up.

Common codes for success and error

Status code	Meaning
200 OK	The server successfully carried out the action that the client requested. For GET request, the response body contains a representation of the requested resource.
204 No content	The server successfully carried out the action, but declined to return any representation. In Ajax, this usually means the browser should not refresh the user interface.
400 Bad request	Generic client-side error. Probably a request format error.
404 Not found	The server cannot find the resource at the requested URL.
500 Internal server error	Generic server-side error. Probably a server-side program run-time error.
503 Service unavailable	The web server is not available, probably because of overloading or maintenance.

Content-Type

- Indicates the data type (MIME) and character encoding of the message body in requests and responses
  - Omitted if empty body
  - Content-Type: application/octet-stream if server cannot decide

HTTP request to login mail.com

```
POST /accounts/auth?service=mail HTTP/1.1
Host: www.mail.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 194

...&email=username&passwd=password&...
```

Notice how character encoding is specified.

HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 12800

<html>...</html>
```

SUSTech
Southern University of Science and Technology

Content-Length

- Indicates the size (in bytes) of the message body in requests and responses
  - Content-length header is sent before the message body
  - Difficult to determine message size if the response is generated dynamically by a program
  - A solution: chunked-transfer encoding (to be discussed)

HTTP request to login mail.com

```
POST /accounts/auth?service=mail HTTP/1.1
Host: www.mail.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 194

...&email=username&passwd=password&...
```

HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 12800

<html>...</html>
```

SUSTech

Common Headers

Header	Request	Response
Host:	Domain name of requested URL	
User-agent:	Which web browser?	
Server:		Which web server?
Referer:	From which web page does the browser obtain this URL?	
Content-type:	Type of content in the body	
Content-length:	Length of the body in bytes	

User-Agent: & Server:

- Indicate which web browser generates the request and which web server generates the response
  - May be used for collecting statistics about web browser market share
  - A server may generate browser specific response

```
GET /test.html HTTP/1.1
Host: httpbin.org
User-Agent: Mozilla 5.0 (Windows)
...
```

request

Browser and server names in actual use are much longer.

```
HTTP/1.1 200 Ok
Server: nginx/1.10
...
```

response

Methods in HTTP/1.0

- HTTP methods are operations on resources
- HTTP/1.0 supports three methods
  - GET – to retrieve a representation of a resource
  - HEAD – to retrieve only the metadata of a resource
  - POST – to submit some data to a resource (program) in web server for processing

GET Method

- **GET** retrieves a representation of a resource
  - Usually empty body in request message
  - Browsers mainly use this method to retrieve web resources
  - Usually a read-only operation

```
GET /a.txt HTTP/1.1
Host: example.com
```

request

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5

Hello
```

response

Head method

- **HEAD** is similar to **GET**, but the response only contains headers and an empty body
- Useful to check the characteristics of a resource without retrieving it, e.g.
  - What is its size?
  - Is the resource still available?

```
HEAD/a.txt HTTP/1.1
Host: example.com
```

request

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5
```

response

POST method

- **POST** submits some data to a resource for processing
  - The resource is usually a server-side program
  - The data are usually encoded in the body of the POST request
  - Usually invokes some server-side action

```
POST /accounts/auth?service=mail HTTP/1.1
Host: www.mail.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 194

...&email=username&passwd=password&...
```

HTTP request to login mail.com

Observation

- The **GET** request in form submission can be …
  - saved in bookmark
  - saved in browser history (possible security problem!)
  - used in a hyperlink <a href="..">. When users click the link, the browser sends the same GET request as in submitting the HTML form.

```
GET /login.php?user=philip&passwd=12345 HTTP/1.1
Host: example.com
```

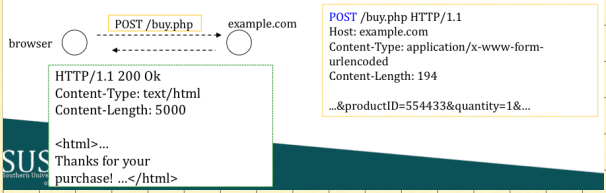
Request for
http://example.com/login.php?user=philip&passwd=12345

Side-effect

- **GET** and **HEAD** have no side-effect
  - read-only operations in web servers
  - A browser may repeat sending these requests without user confirmation
  - May repeat a **GET** request in case of network error
- **POST** may have side-effect
  - May cause some actions in the server (e.g. adding a record, sending an email, placing an order)
  - A browser must confirm with users before resending such requests

resubmit a POST request

- In this example, a purchase request is sent as a POST (because it has side-effect).
- The server returns the transaction result in the response
- If the user refreshes the result page, the browser would need to send the POST request twice.
  - What would happen?



Confirmation when resubmitting request

- Because GET causes no action in the server, it should make no difference to repeat or skip a GET request
  - The browser may resubmit a GET request
- POST may have side-effect. If the browser resubmits a POST request, the server may perform some action twice.
  - The browser must confirm with users before resubmitting a POST request
  - *Users may be confused by the resend warning … Solution: Post/Redirect/Get pattern*

Comparison: GET and POST

	GET	POST
Typical usage	Readonly query like Google search, Google Charts	Request that triggers action on server side. E.g. place an order, login, save a file
Side-effect?	Read-only operations in servers. Browsers can resubmit requests without confirmation	May cause write operations in servers. Browsers must confirm with users.
Form data in URL?	Yes. The query URL can be saved in bookmark and hyperlinks.	No
Support file upload	No	Yes

cURL: Utility for URL

- cURL is a computer software project providing a library and command-line tool for transferring data using various protocols.
  Download: https://curl.haxx.se/download.html
- Example:

```
$ curl http://httpbin.org/headers
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.55.1"
    "X-Amzn-Trace-Id": "Root=1-5f646cba-4f1892177757747d3883a5ed"
  }
}
```

cURL: Inspect HTTP transaction

```
$ curl http://httpbin.org/headers --HEAD -v
> HEAD /headers HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.55.1
> Accept: */*
```

```
...
< HTTP/1.1 200 OK
< Server: gunicorn/19.9.0
< Date:  Fri, 17 Sep 2021 14:55:18 GMT
< Content-Type: application/json
< Content-Length: 173
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
```

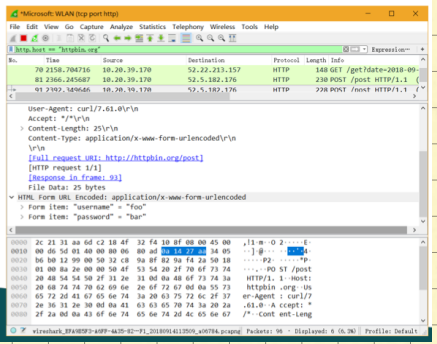
cURL: Inspect GET request

```
$ curl http://httpbin.org/get
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.55.1",
    "X-Amzn-Trace-Id": "Root=1-6140670b-03d479b60088bb886a908dcd"
  },
  "origin": "116.6.234.150",
  "url": "http://httpbin.org/get"
}
```

```
$ curl http://httpbin.org/get?date=2021-09-18
{
  "args": {
    "date": "2021-09-18"
  },
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.55.1",
    "X-Amzn-Trace-Id": "Root=1-614066d5-7e918fbc5ca540b9583f976b"
  },
  "origin": "116.6.234.150",
  "url": "http://httpbin.org/get?date=2021-09-18"
}
```

cURL : Inspect POST request

```
$ curl -d "username=foo&password=bar" -X POST http://httpbin.org/post
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "password": "bar",
    "username": "foo"
  },
  "headers": {
    ...
}
```



STech
University of Science and Technology