SEC201.2 Web-Based Programming

What is Internet?

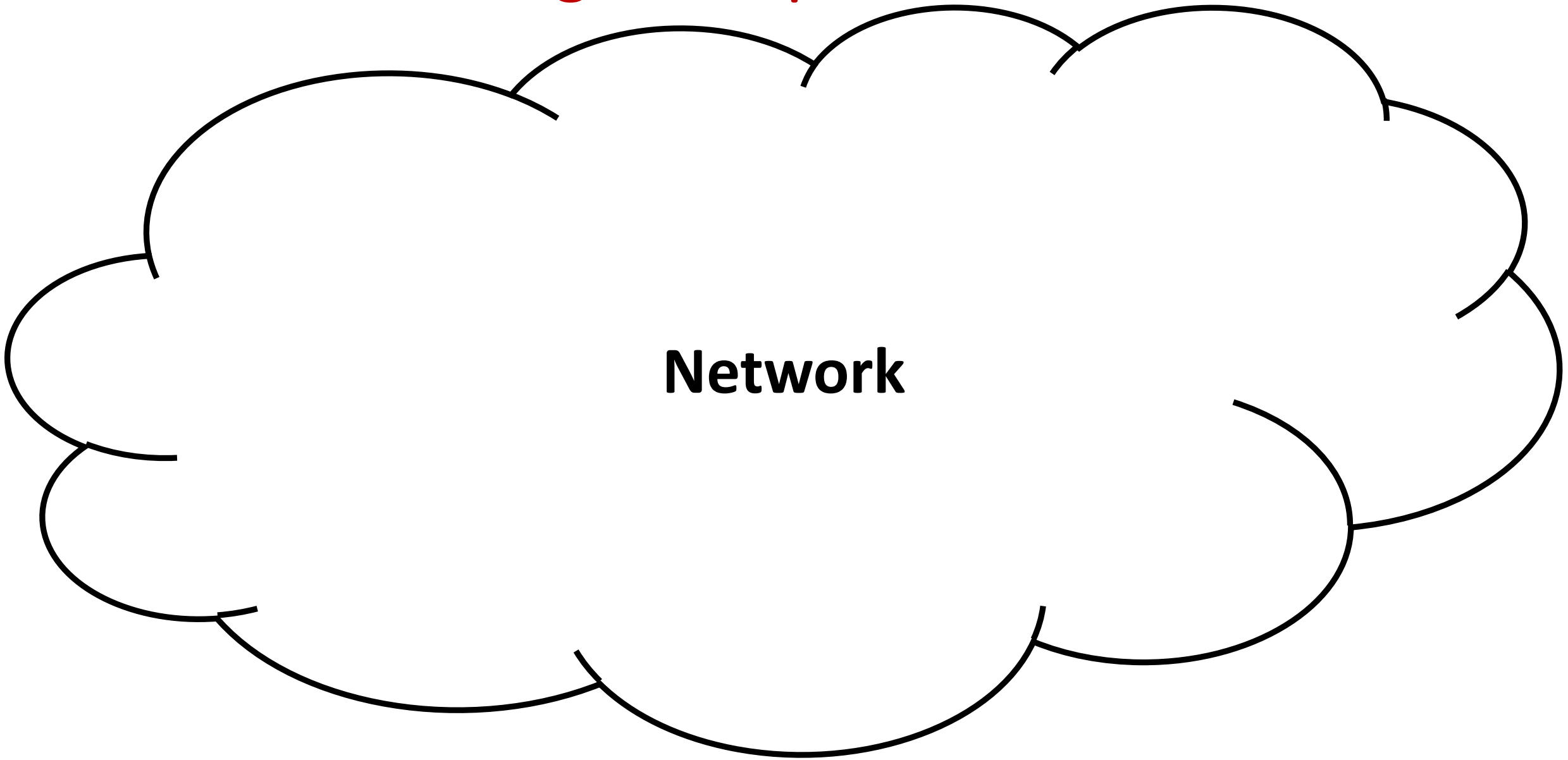
How does the Internet Work?

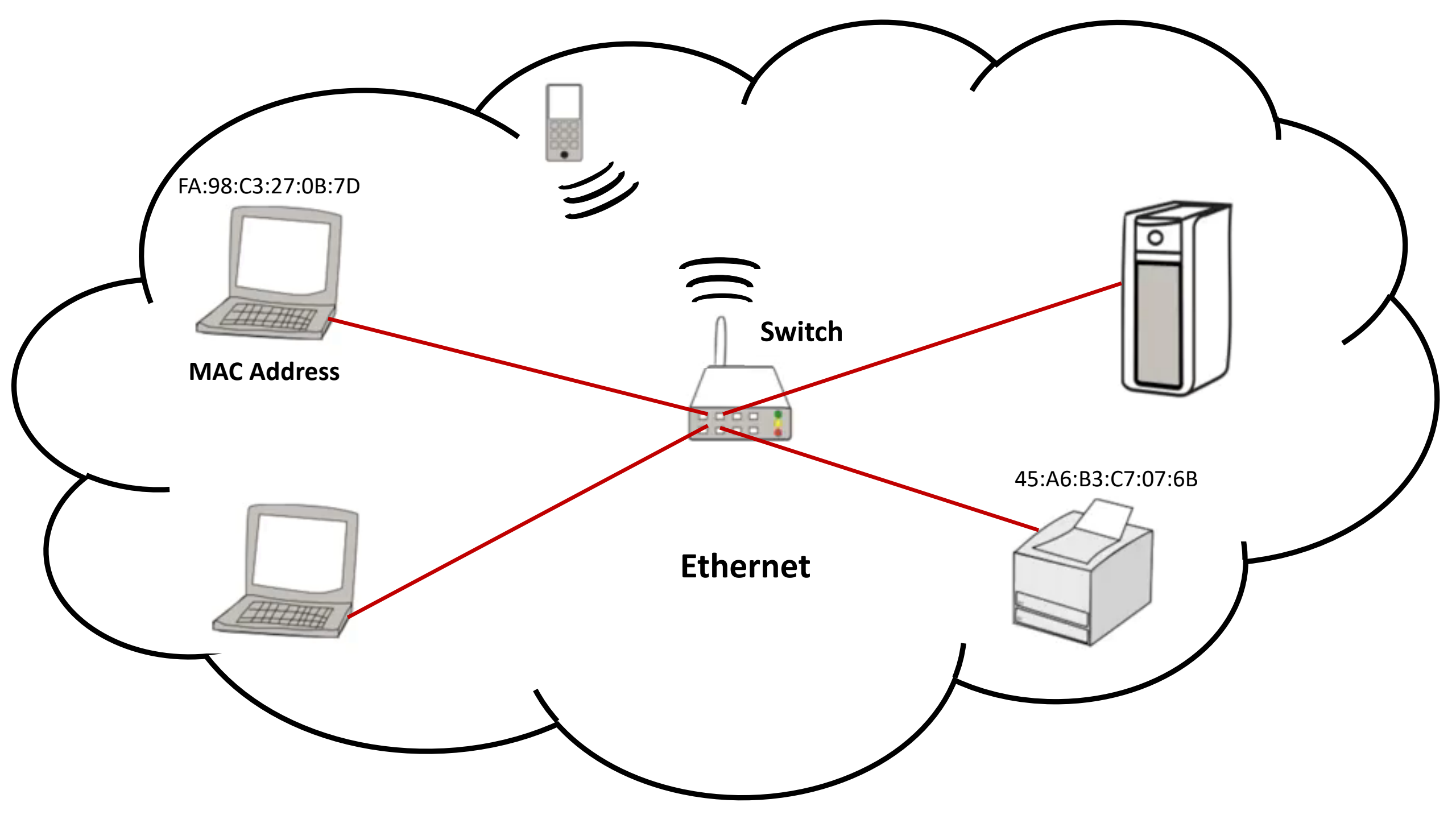
Outline

- How the Internet Works: An Overview
  - Networking Concepts
  - The Internet
  - Internet Hot Topics

- The HTTP Protocol
  - HTTP Overview
  - HTTP Request
  - HTTP Response
  - HTTP Sessions and Cookies

Basic Networking Concepts



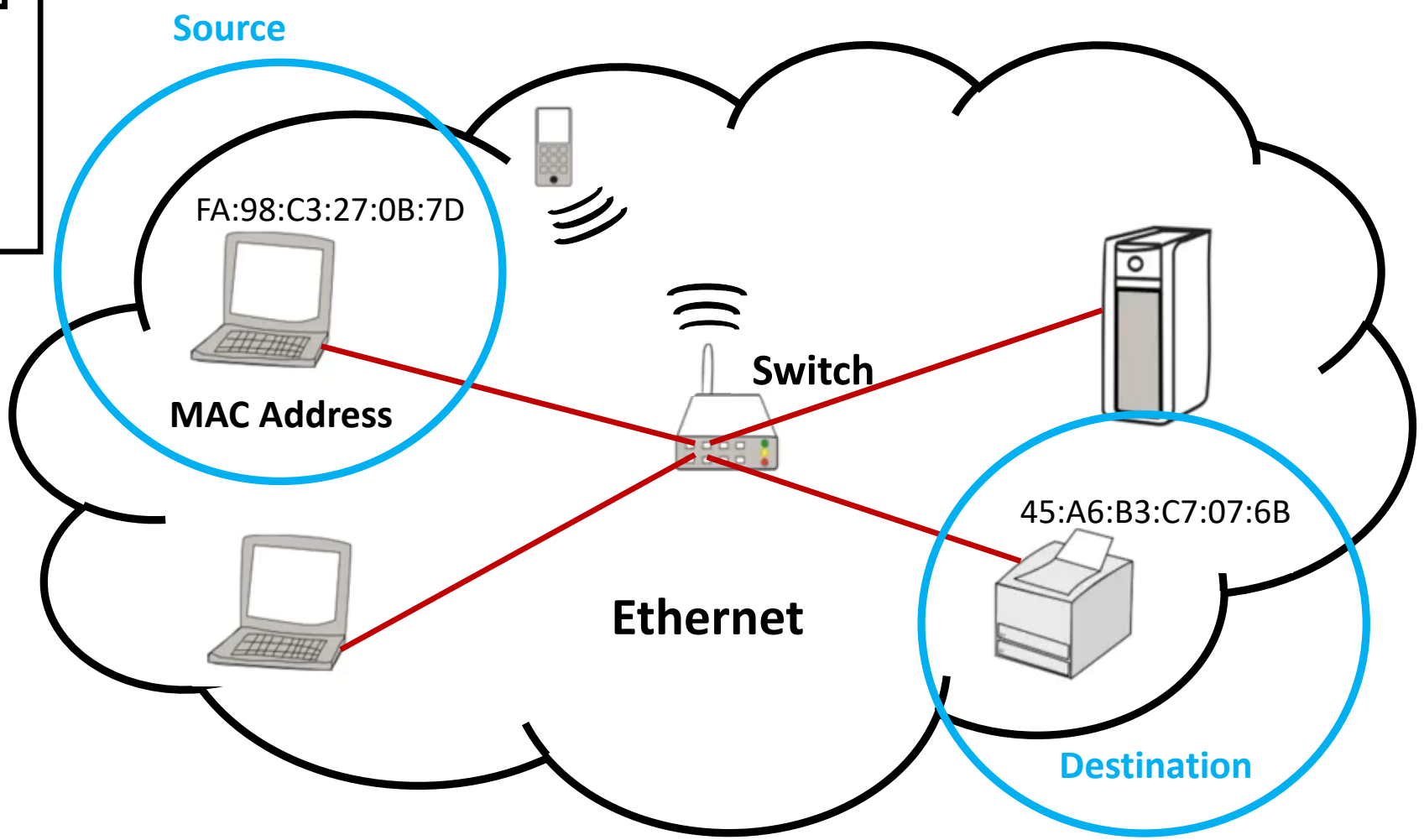

FA:98:C3:27:0B:7D
MAC Address
Switch
45:A6:B3:C7:07:6B
Ethernet

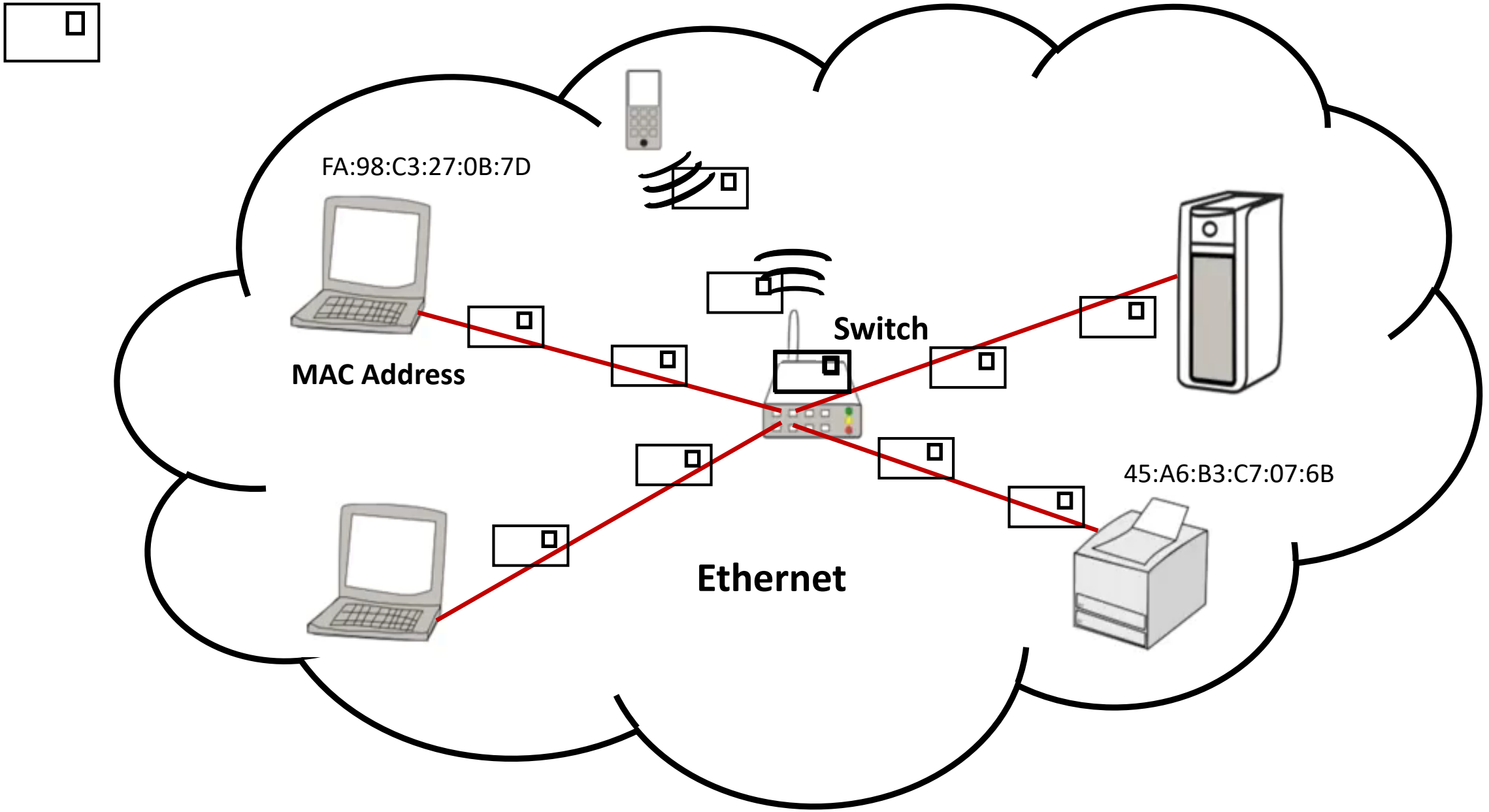
Frame

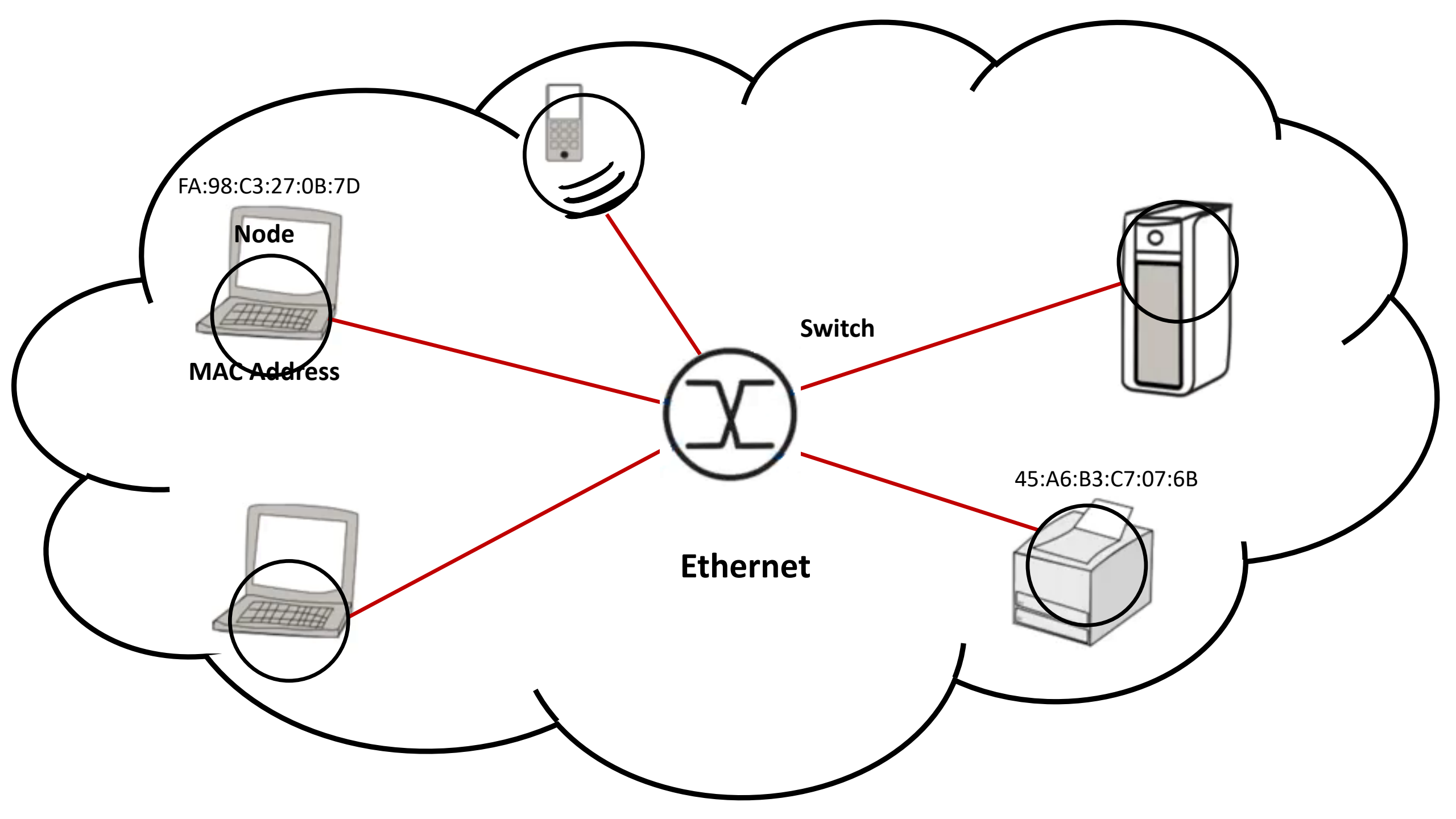
Source: FA:98:C3:27:0B:7D

Destination: 45:A6:B3:C7:07:6B

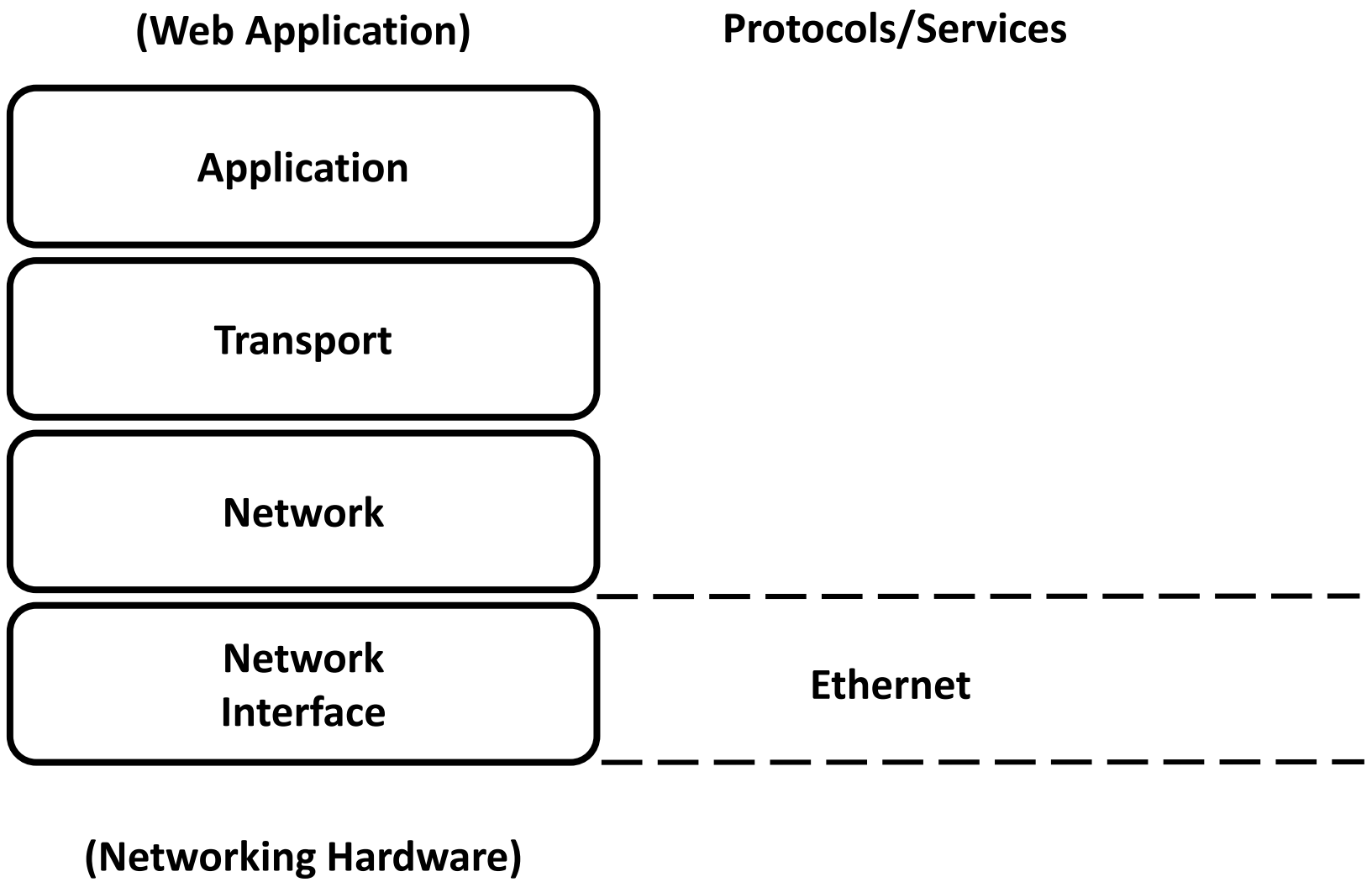
Data




FA:98:C3:27:0B:7D
MAC Address
Switch
45:A6:B3:C7:07:6B
Ethernet


FA:98:C3:27:0B:7D
Node
MAC Address
Switch
45:A6:B3:C7:07:6B
Ethernet

TCP/IP
Networking Model

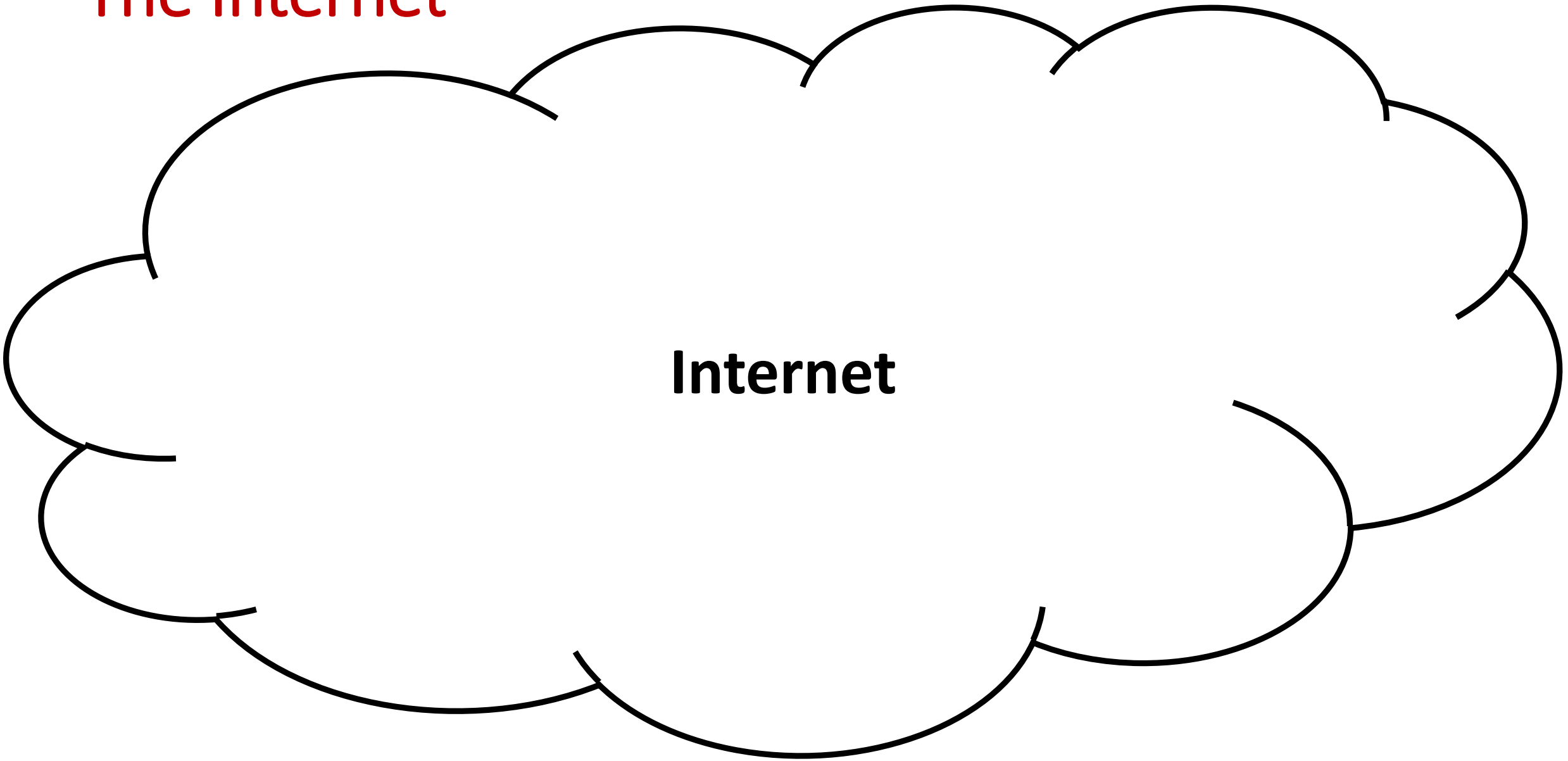


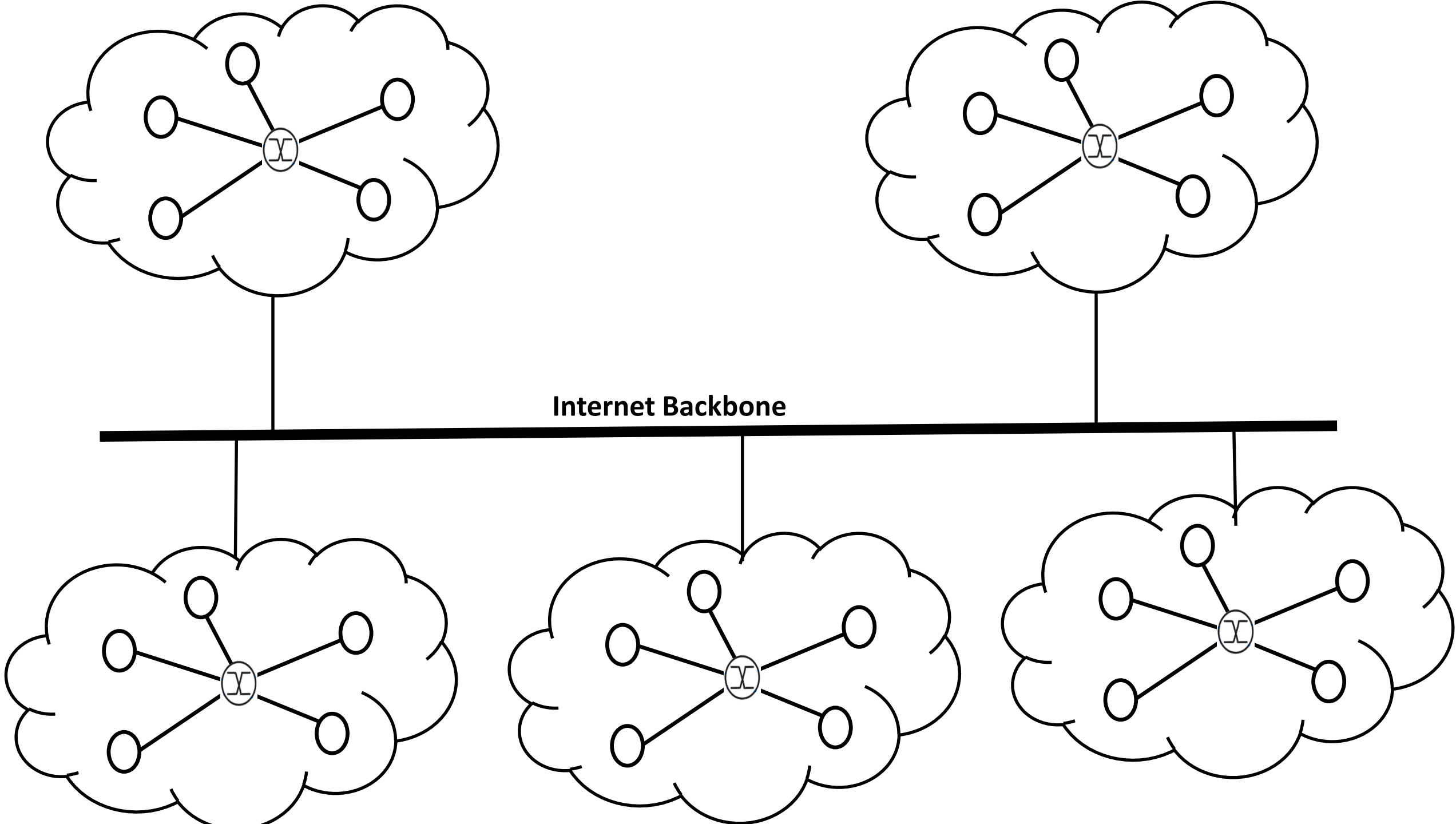
Outline

- **How the Internet Works: An Overview**
  - Networking Concepts
  - **The Internet**
  - Internet Hot Topics

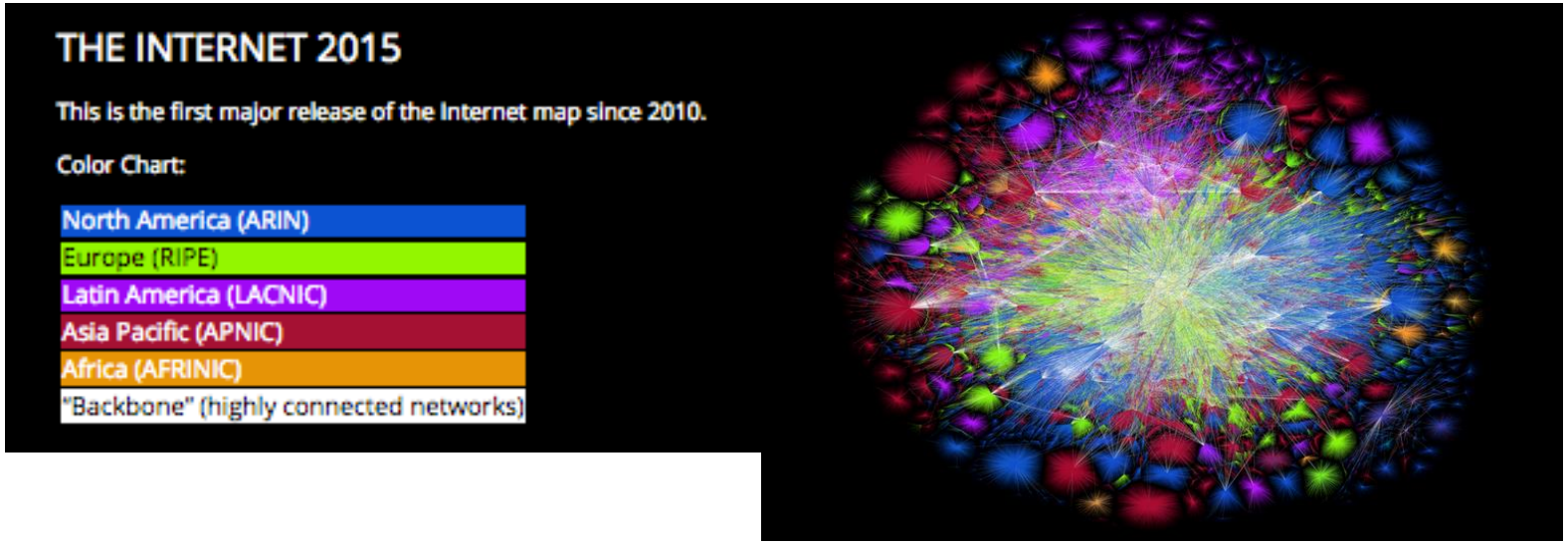
- The HTTP Protocol
  - HTTP Overview
  - HTTP Request
  - HTTP Response
  - HTTP Sessions and Cookies

The Internet




Internet Backbone

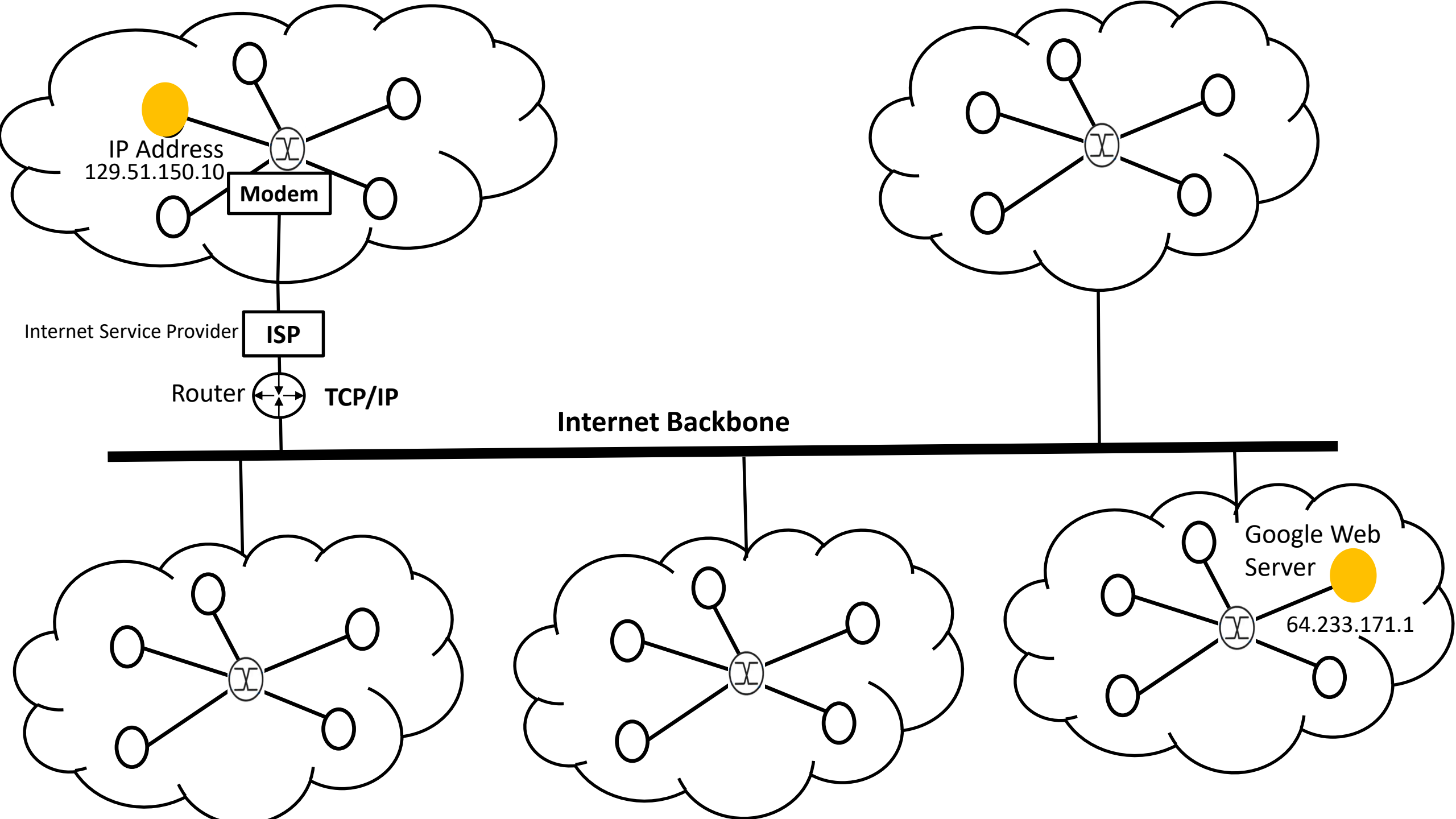
Internet Backbone



https://time.com/3952373/internet-opte-project/

THE INTERNET 2021

https://www.opte.org/the-internet

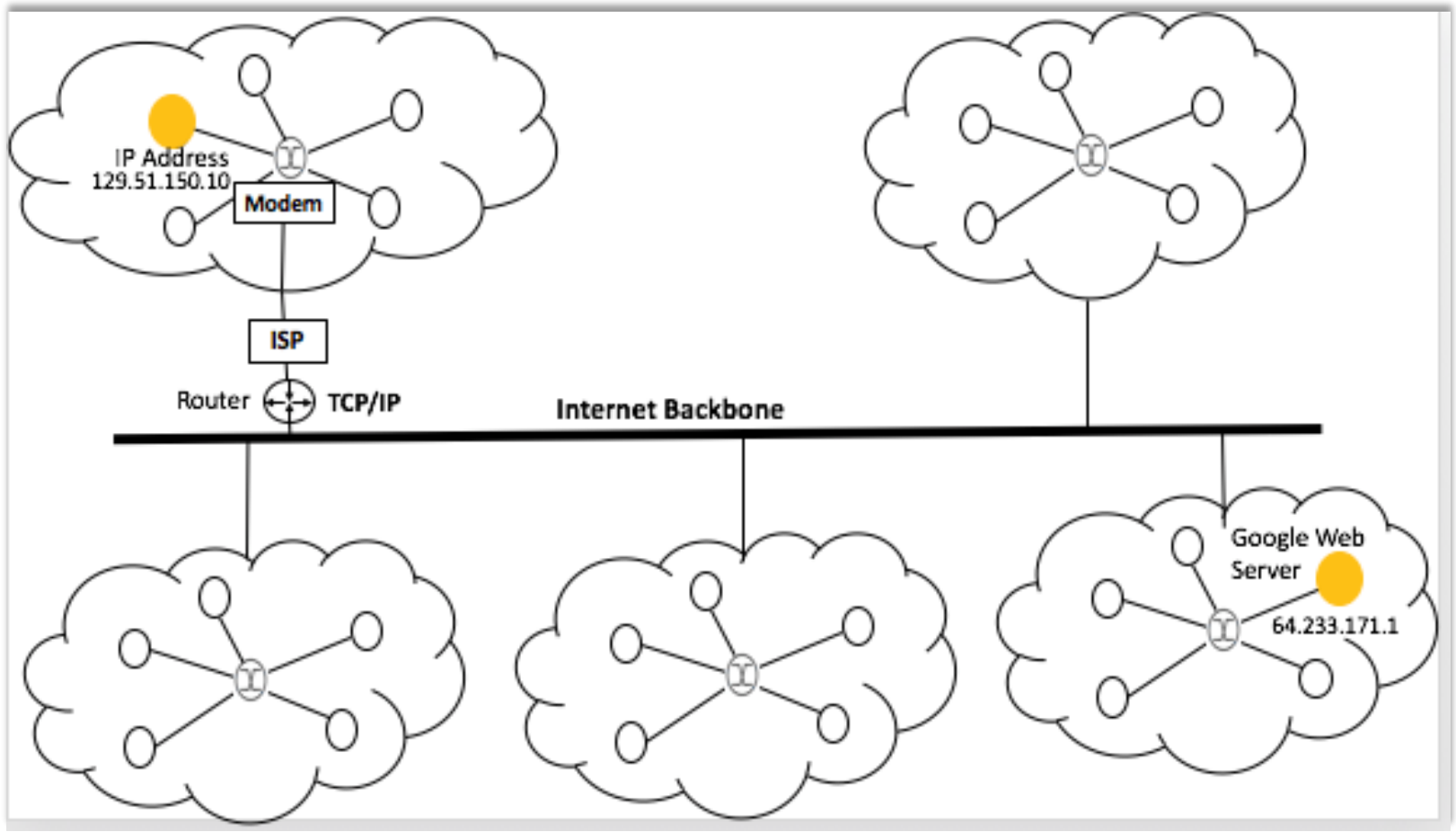

IP Address
129.51.150.10
Modem
Internet Service Provider
ISP
Router
TCP/IP
Internet Backbone
Google Web
Server
64.233.171.1

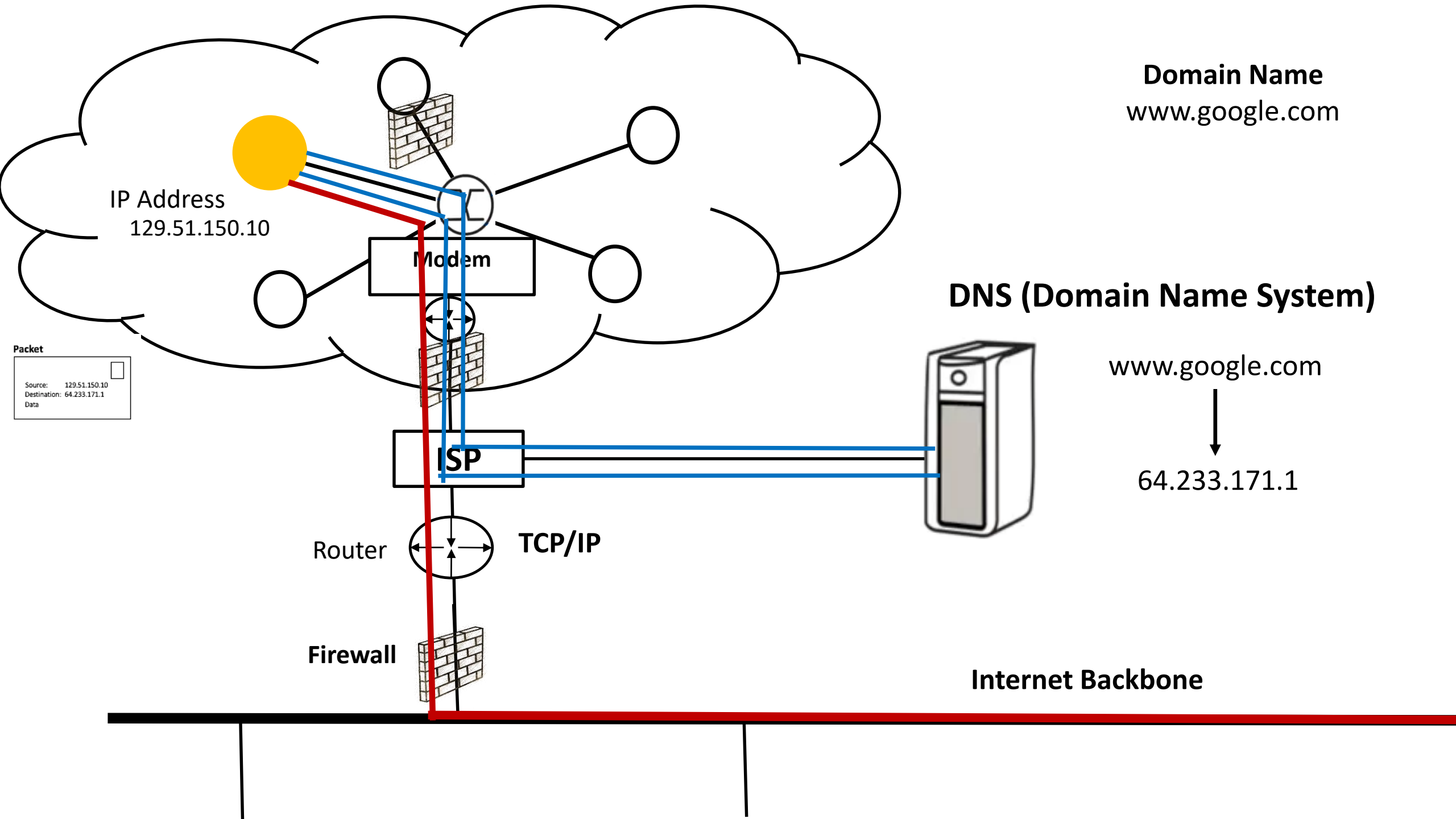
Packet

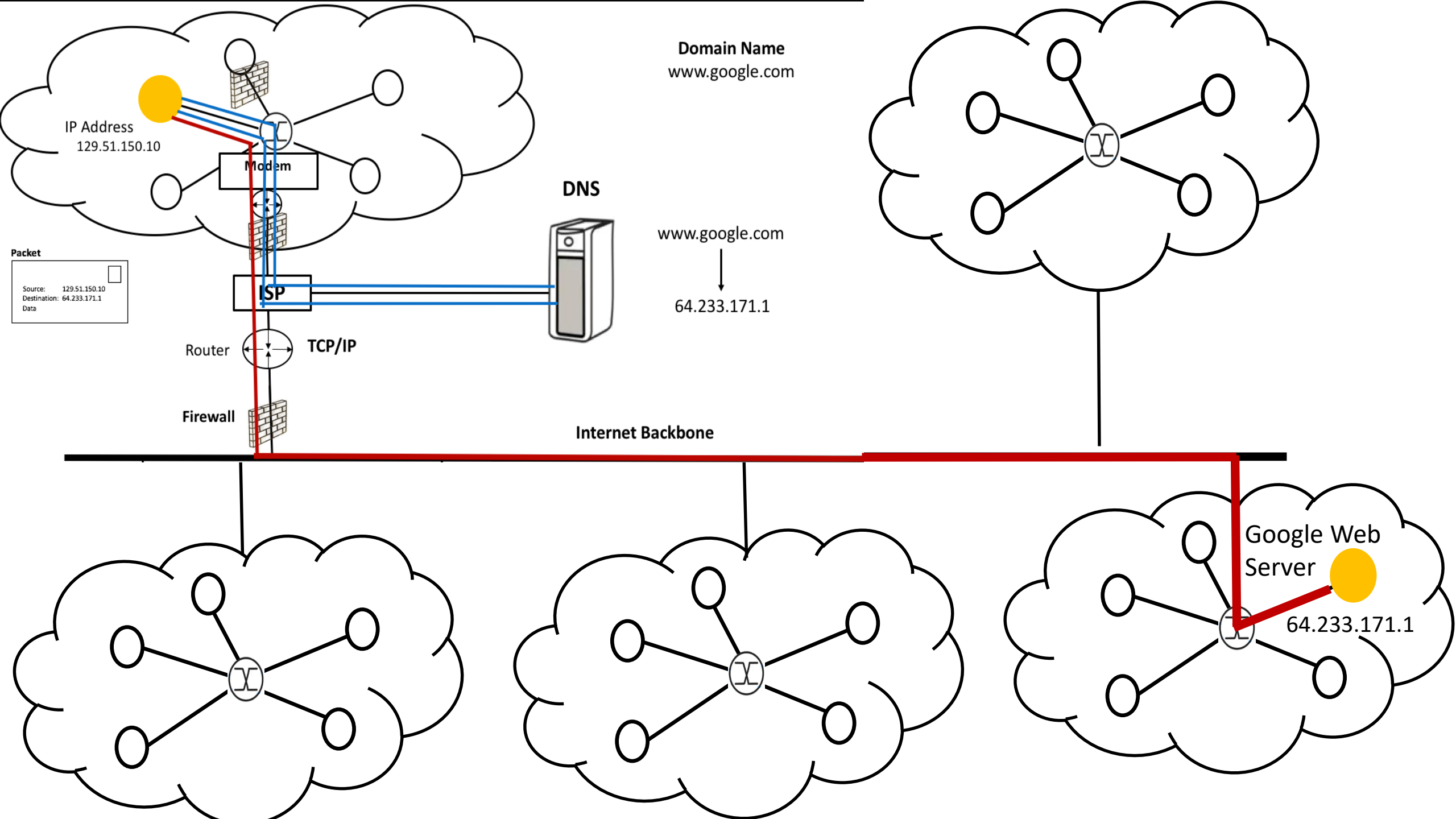
Source: 129.51.150.10

Destination: 64.233.171.1

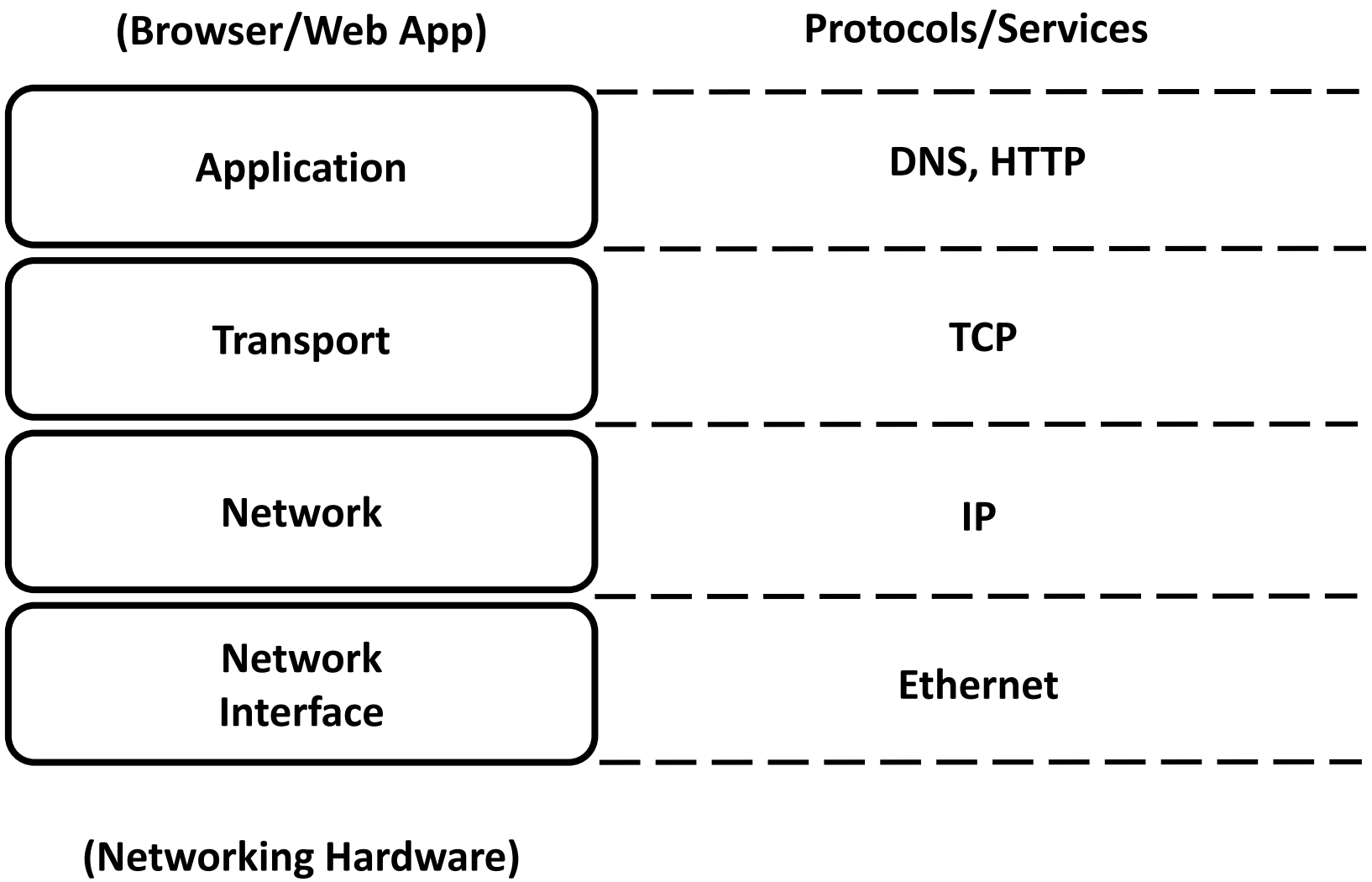
Data



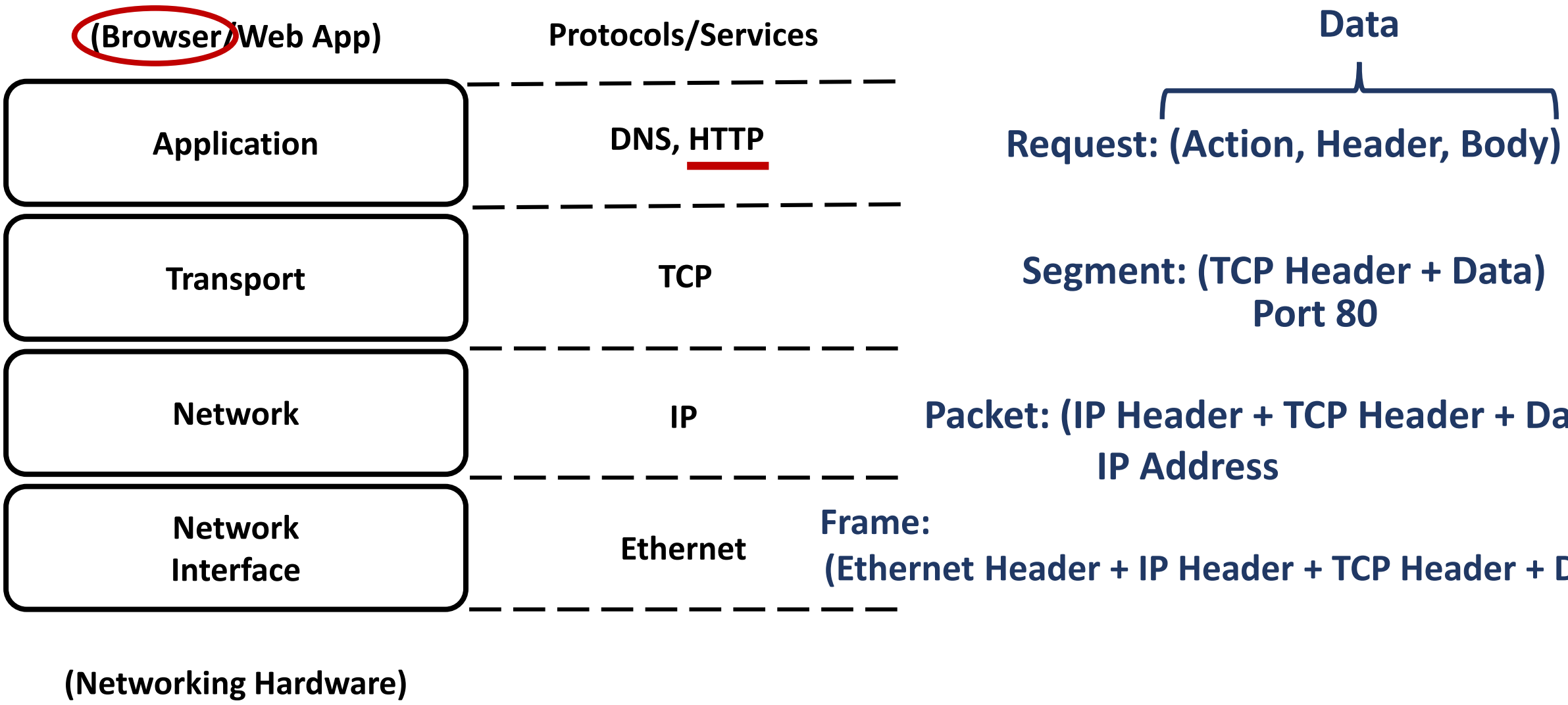

Domain Name
www.google.com
IP Address
129.51.150.10
Modem
DNS (Domain Name System)
Packet
Source: 129.51.150.10
Destination: 64.233.171.1
Data
www.google.com
ISP
64.233.171.1
Router
TCP/IP
Firewall
Internet Backbone


Domain Name
www.google.com
IP Address
129.51.150.10
Modem
DNS
www.google.com
64.233.171.1
Packet
Source: 129.51.150.10
Destination: 64.233.171.1
Data
ISP
Router
TCP/IP
Firewall
Internet Backbone
Google Web
Server
64.233.171.1

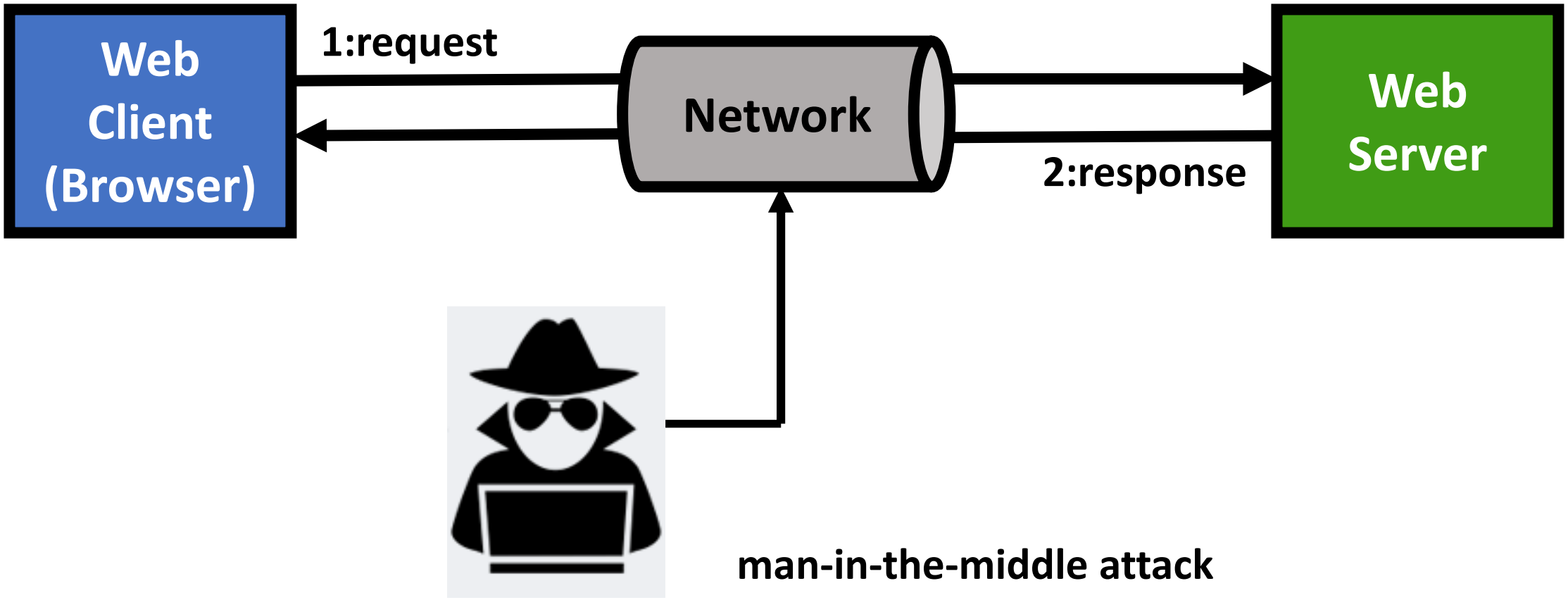
TCP/IP
Networking Model



TCP/IP Networking Model

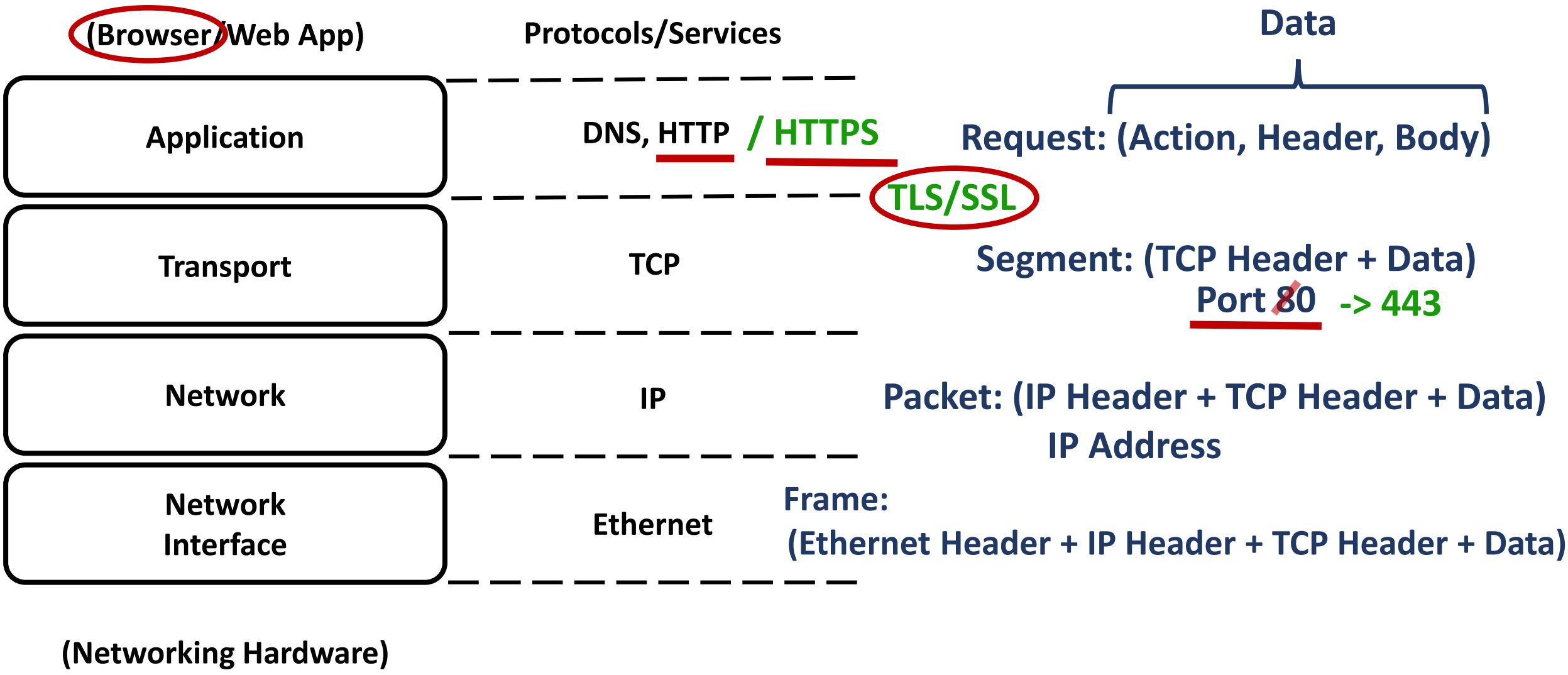


HTTP Protocol



man-in-the-middle attack

TCP/IP Networking Model



Who is deciding the Ports?

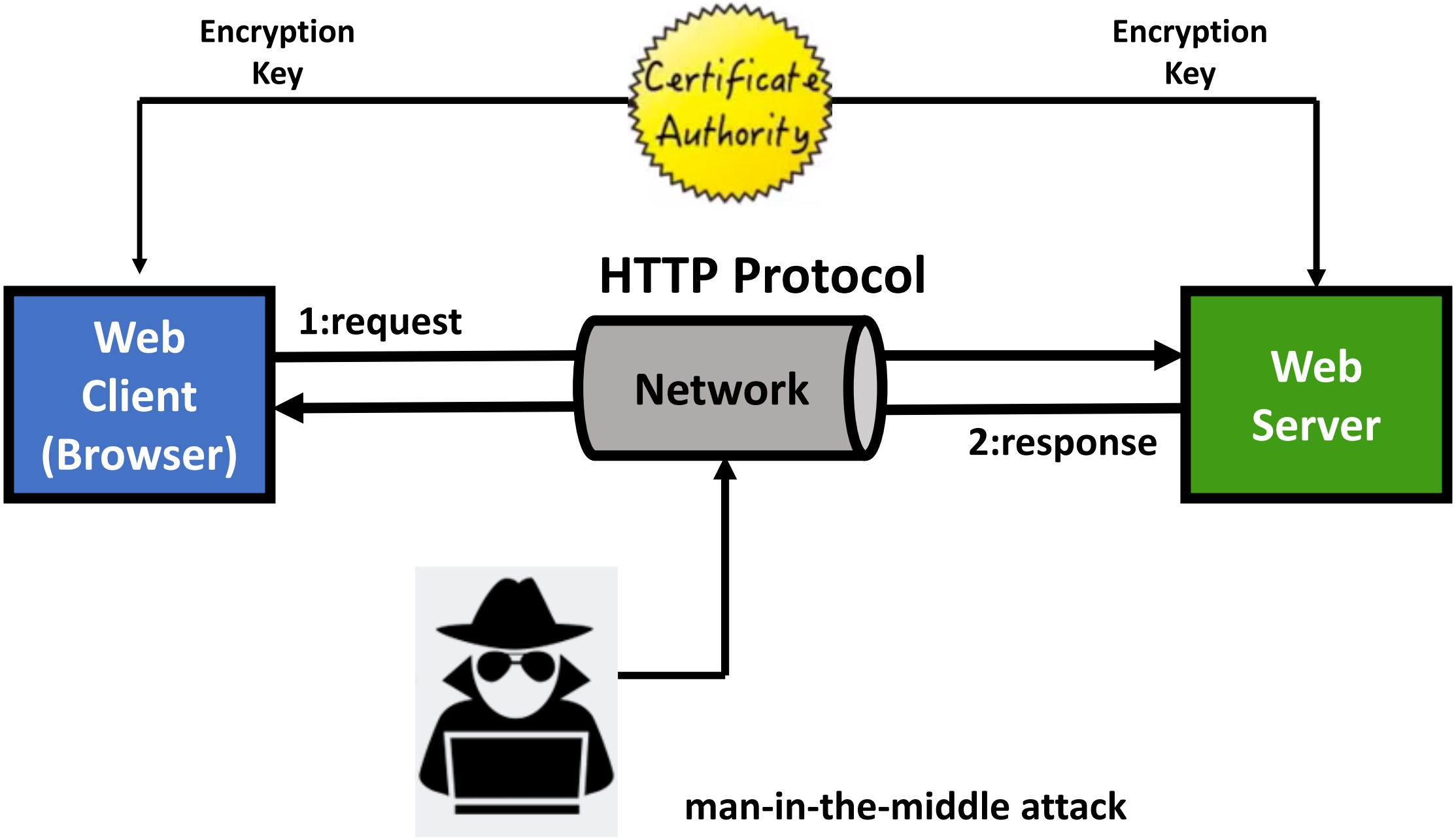
Internet Engineering Task Force (IETF)

Internet Corporation for Assigned Names and Numbers (ICANN)

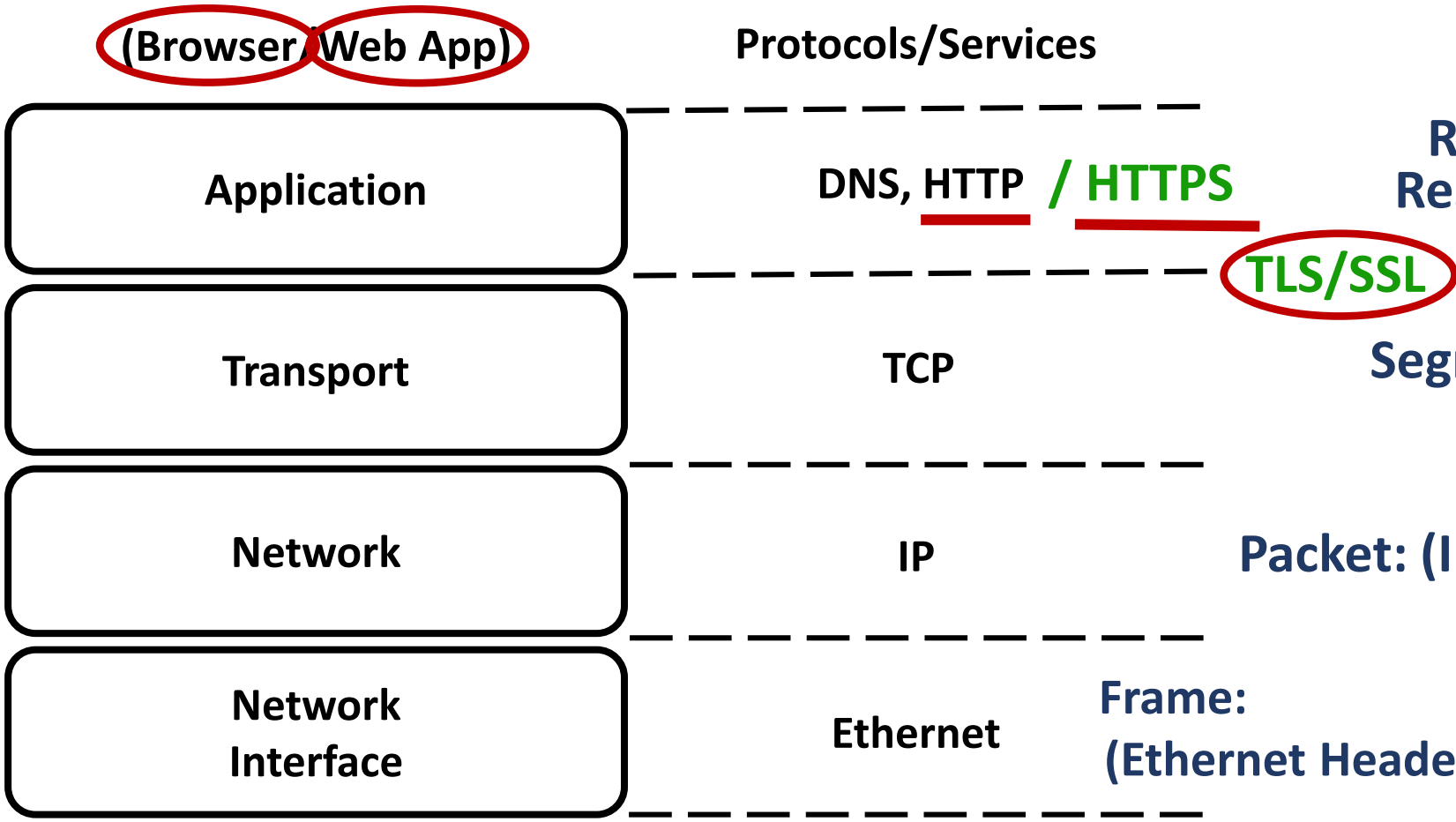
Internet Assigned Numbers Authority (IANA)

HTTP → Port 80

HTTPS → Port 443


Encryption
Key
Certificate
Authority
Encryption
Key
HTTP Protocol
Web
Client
(Browser)
1:request
Network
Web
Server
2:response
man-in-the-middle attack

TCP/IP Networking Model



Data

Request: (Action, Header, Body)
Response: (Status, Header, Body)

Segment: (TCP Header + Data)
Port ~~80~~ -> 443

Packet: (IP Header + TCP Header + Data)
IP Address

Frame:
(Ethernet Header + IP Header + TCP Header + Data)

Outline

- **How the Internet Works: An Overview**
  - Networking Concepts
  - The Internet
  - **Internet Hot Topics**

- The HTTP Protocol
  - HTTP Overview
  - HTTP Request
  - HTTP Response
  - HTTP Sessions and Cookies

Internet Hot Topics

- Net Neutrality
  - The principle that all Internet traffic should be treated equally
- Security and Privacy
  - With more and more valuable information available via Internet banking, credit card transactions, Facebook information, how do we secure private data?
- Cloud Computing
  - A type of Internet based computing where shared resources, that is computing storage or services are provided on demand
- Big Data and Analytics
  - With the massive amount of data being collected (including IoTs), we don’t have ability to analyze data and discover new facts

Outline

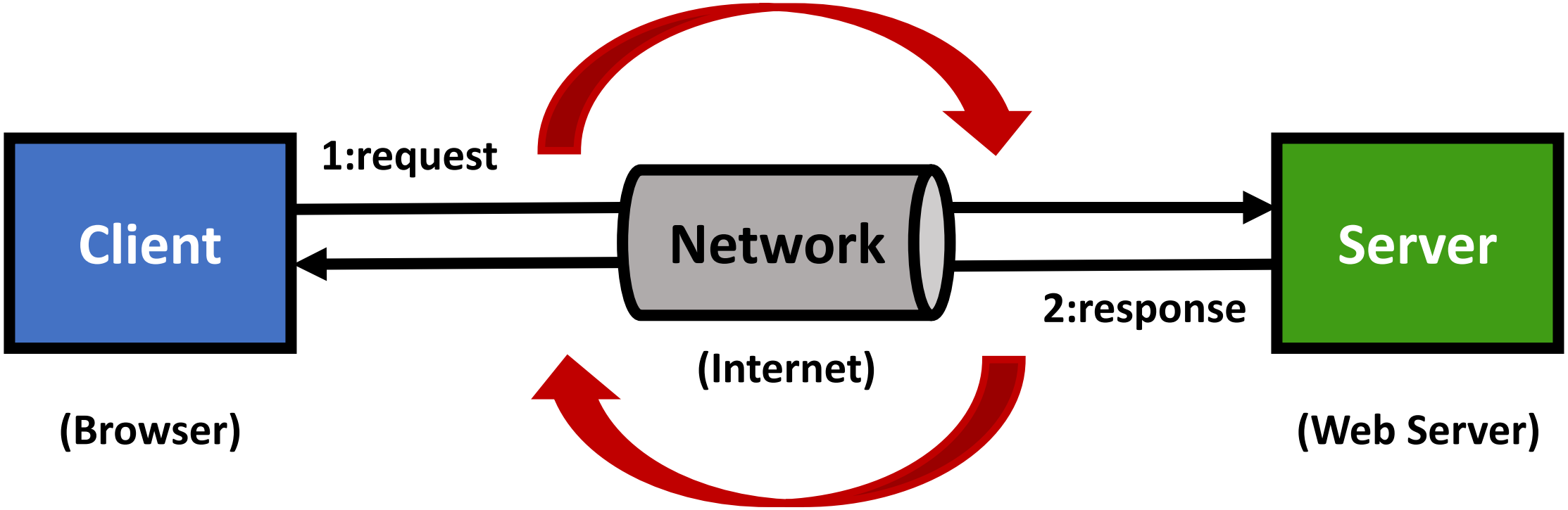
- How the Internet Works: An Overview
  - Networking Concepts
  - The Internet
  - Internet Hot Topics

- **The HTTP Protocol**
  - **HTTP Overview**
  - HTTP Request
  - HTTP Response
  - HTTP Sessions and Cookies

Review: Web Apps – Architecture (Model)

- Client-Server Architecture – the most basic model for describing the relationship between the cooperating programs in a networked software application
- Client-Server Architecture consists of two parts:
  1. **Server** – "listens" for requests, and provides services and/or resources according to those requests
  2. **Client** – establishes a connection to the server, and requests services and/or resources from the server

Hypertext Transfer Protocol (HTTP)



Request – Response Cycle

HTTP Protocol

- Used to deliver resources in distributed hypermedia information systems

- In order to build and debug web applications, it’s vital to have a good understanding of how HTTP works

HTTP Resources

- **Hypertext** – Text, marked up using HyperText Markup Language (HTML), possibly styled with CSS, and containing references (i.e., hyperlinks) to other resources
- **Hypermedia** – The logical extension of hypertext to graphics, audio and video
- **Hyperlinks** – Define a structure over the Web
- **Scripts** – Code that can be executed on the client side

HTTP Basics

HTTP has always been a **stateless protocol**

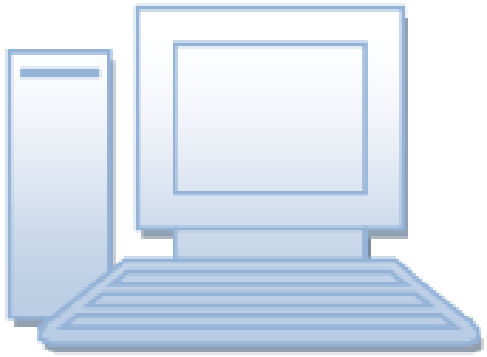
- Server not required to retain information related to previous client requests
- Each client request is executed independently, without any knowledge of the client requests that preceded it
- Difficult for web applications to respond intelligently to user input, i.e., to create the interactivity that users expect
- Cookies, sessions, URL encoded parameters and a few other technologies have been introduced to address this issue

Outline

- How the Internet Works: An Overview
  - Networking Concepts
  - The Internet
  - Internet Hot Topics

- **The HTTP Protocol**
  - HTTP Overview
  - **HTTP Request**
  - HTTP Response
  - HTTP Sessions and Cookies

(1) User issues URL from a browser
http://host:port/path/file



(5) Browser formats the response and displays

Client (Browser)

(2) Browser sends a request message

```
GET URL HTTP/1.1
Host: host:port
.................
.................
```

(4) Server returns a response message

```
HTTP/1.1 200 OK
.................
.................
.................
```

HTTP (Over TCP/IP)

(3) Server maps the *URL* to a file or program under the document directory.

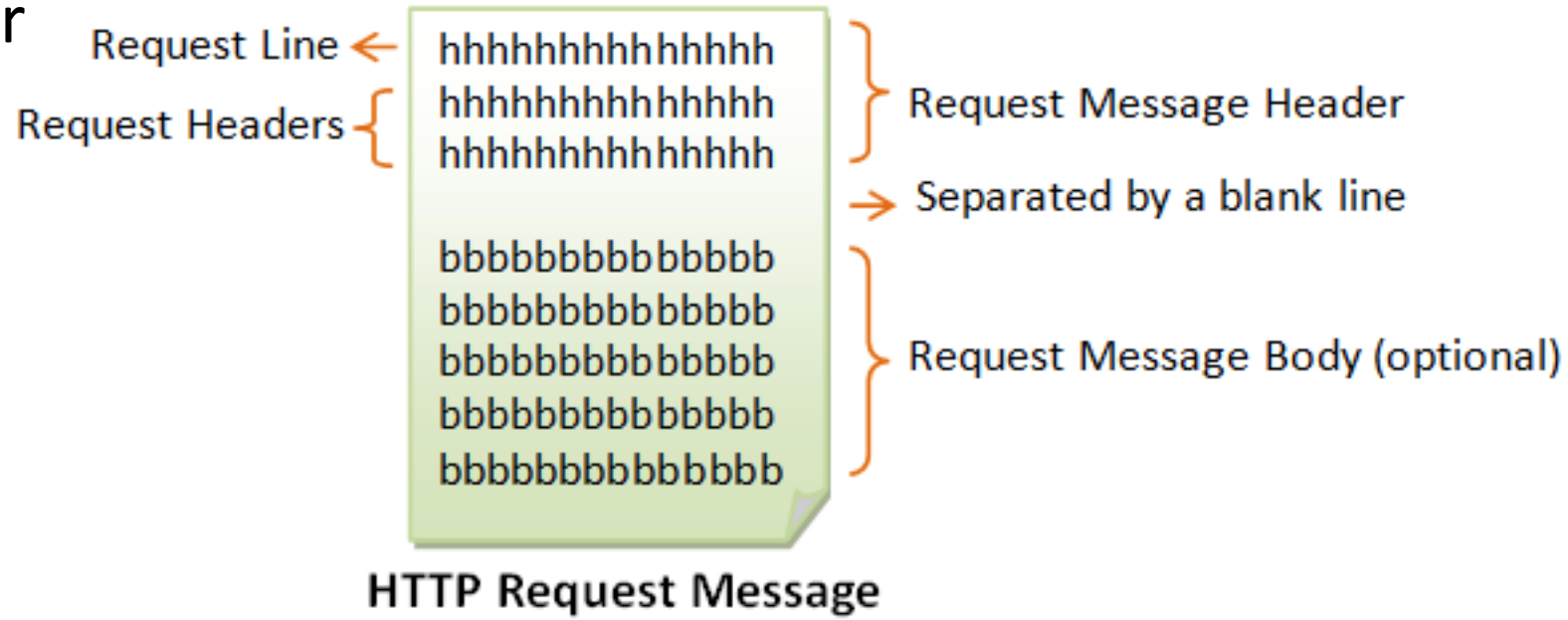
Server (@ *host:port*)

HTTP Request

HTTP – Client Side

An HTTP **request message** consists of three parts:

1) Request line
2) Request Header
3) Message body



(1) HTTP – Client Side: Request Line

- The request line identifies the resource, and the desired action ("request" or "method") that should be applied to it.

- The request line has the following syntax:

  request-line-method request-URI HTTP-version

  - ***Request-line-method:*** HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS, and the client can use one of these methods to send a request to the server
  - ***Request-URI*:** specifies the resource requested
  - ***HTTP-version:*** specifies the version of the HTTP

- Examples of request line are:

```
GET /test.html HTTP/1.1
HEAD /query.html HTTP/1.0
POST /index.html HTTP/1.1
```

HTTP – Client Side: Request Line Methods

1. **GET** – get a web resource from the server

2. **HEAD** – get the header that a GET request would have obtained, but without the response body. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy

3. **POST** – post/submit data (e.g., from an HTML form) to the web server, where the data is supplied in the body of the request, and the result may be the creation of a new resource, or the update of an existing one

4. **PUT** – ask the server to store the data

HTTP – Client Side: Request Line Methods

5. **DELETE** – ask the server to delete the data
6. **TRACE** – Echoes back the received requested
7. **OPTIONS** – returns the HTTP methods that the server supports for the specified resource
8. **CONNECT** – used to tell a Proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it (usually to facilitate SSL through HTTPS)
9. **PATCH** – apply partial modifications to a resource

HTTP – Client Side: Request Line Methods

- HEAD, GET, OPTIONS and TRACE are referred to as **safe methods**
- Safe methods should not produce side effects on the server
- **Note:** If a GET method is implemented in a safe way, a browser can make arbitrary Get requests without modifying the state of a web application, i.e., these requests can be cached

HTTP – Client Side: Request Line Methods

- The POST, PUT, and DELETE methods may cause side effects on the server – they are not considered safe
- Furthermore, the PUT and DELETE methods should be **idempotent** – multiple identical requests should have the same effect as a single request
- Note that safe methods, since they don't change the state of the server, are idempotent

HTTP – Client Side: Request Line URI

- The resource is typically identified by a Universal Resource Identifier (URI)
- **Note:** a Uniform Resource Locator (URL) is a specific type of URI
- URL has the following syntax:

protocol://hostname:port/path-and-file-name

- There are 4 parts in a URL:
  - ***Protocol***: The application-layer protocol used by the client and server, e.g., HTTP, FTP, and telnet
  - ***Hostname***: The domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server
  - ***Port***: The TCP port number that the server is listening for incoming requests from the clients
  - ***Path-and-file-name***: The name and location of the requested resource, under the server document base directory
- Examples
  - http://www.nowhere.com/docs/index.html
  - ftp://www.ftp.org/docs/test.txt
  - mailto:user@test101.com

(2) HTTP – Client Side: Request Headers

- The HTTP message header is the primary part of an HTTP request
- The request headers are in the form of **name:value** pairs
- Multiple values, separated by commas, can be specified
- Header fields syntax:

field_name: field_value

Ex:

```
Host: www.xyz.com
Connection: keep-alive
Accept: text/plain, image/gif, image/jpeg
Accept-Language: en-us, fr, cn, tr
```

HTTP – Client Side: Request Headers

- Fields may be any application-specific strings, but a core set of fields is standardized by the Internet Engineering Task Force (IETF)

- An HTTP message header must be separated from the message body by a blank line

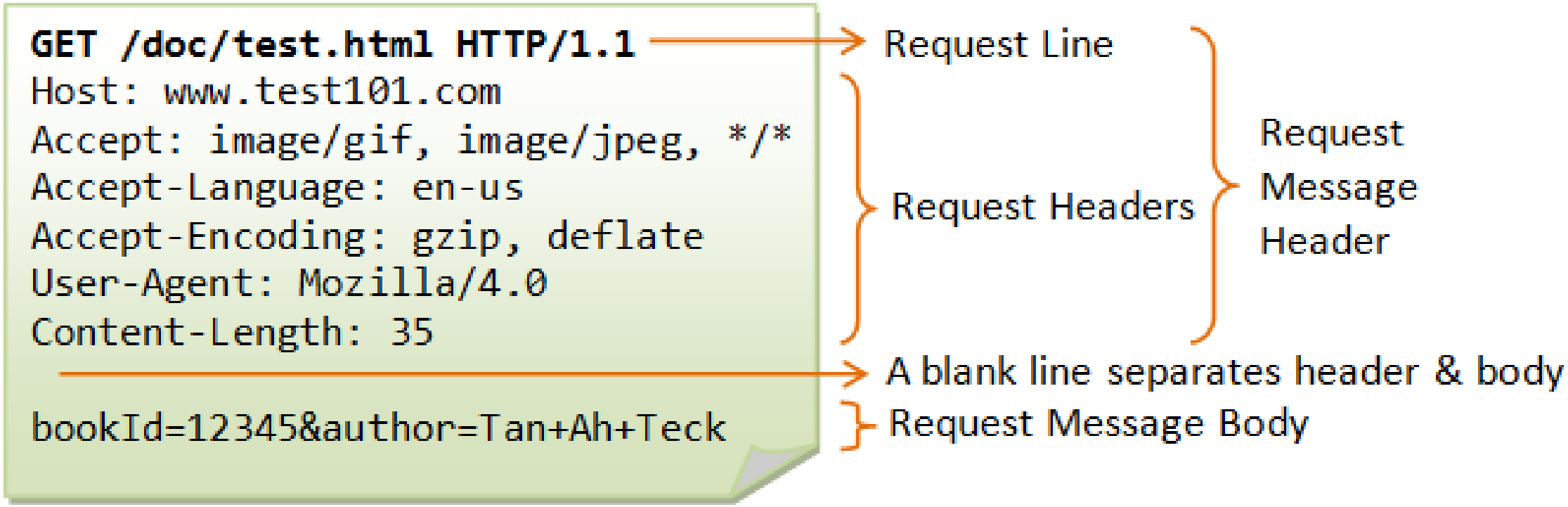
(3) HTTP – Client Side: Message Body

- The **message body** in an HTTP request is optional
- It typically includes user-entered data or files that are being uploaded
- If an HTTP request includes a body, there are usually header fields that describe the body

Ex:

```
Content-Type: text/html
Content-Length: 3495
```

A Sample HTTP Request Message



Outline

- How the Internet Works: An Overview
  - Networking Concepts
  - The Internet
  - Internet Hot Topics

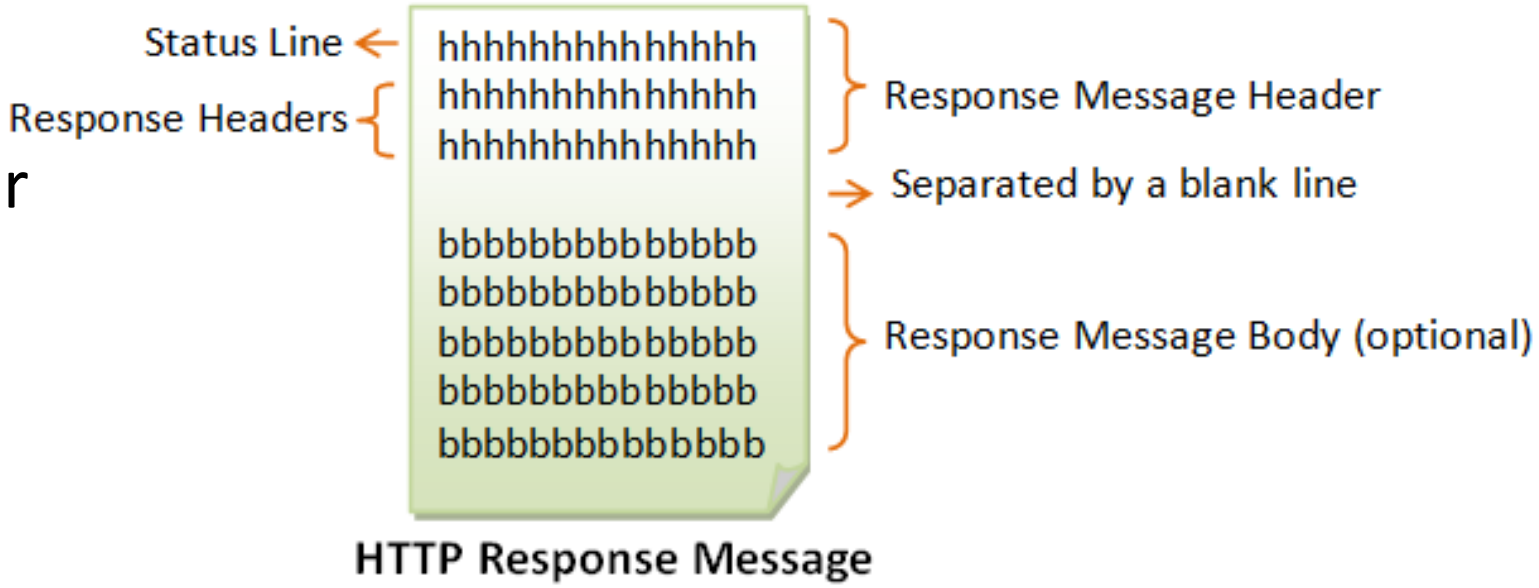
- **The HTTP Protocol**
  - HTTP Overview
  - HTTP Request
  - **HTTP Response**
  - HTTP Sessions and Cookies

HTTP Response

HTTP – Server Side

An HTTP **response message** is similar to a request message – it also consists of three parts:

1) Status line
2) Response Header
3) Message body



HTTP Response Message

(1) HTTP – Response: Status Line

- The **status line** is the first line of the response provided by the server
- The status line has the following syntax:

  HTTP-version status-code reason-phrase

- It consists of three parts:
  1) The **HTTP version**, in the same format as in the message request, e.g., HTTP/1.1
  2) A response **status code** that provides the result of the request (3-digit number)
  3) An English **reason phrase** describing the status code

Ex:

```
HTTP/1.1   200   OK
HTTP/2     404   Not Found
HTTP/1.0   403   Forbidden
```

HTTP – Response: Status Line – Status Code

The **status code** associated with the status line belong to one of five categories:

- **1xx (Provisional Response)** – A provisional response that requires the requestor to take additional action in order to continue
- **2xx (Successful)** – The server successfully processed the request
- **3xx (Redirected)** – Further action is needed to fulfill the request. Often, these status codes are used for redirection
- **4xx (Request Error)** – There was likely an error in the request which prevented the server from being able to process it
- **5xx (Server Error)** – The server had an internal error when trying to process the request

Some Commonly Encountered Status Codes

- **100 Continue**: The server received the request and in the process of giving the response
- **200 OK**: The request is fulfilled
- **301 Move Permanently**: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location
- **400 Bad Request**: Server could not interpret or understand the request, probably syntax error in the request message
- **401 Authentication Required**: The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password)
- **403 Forbidden**: Server refuses to supply the resource, regardless of identity of client
- **404 Not Found**: The requested resource cannot be found in the server
- **405 Method Not Allowed**: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested
- **408 Request Timeout**
- **500 Internal Server Error**: Server is confused, often caused by an error in the server-side program responding to the request
- **501 Method Not Implemented**: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get")
- **502 Bad Gateway**: Proxy or Gateway indicates that it receives a bad response from the upstream server
- **503 Service Unavailable**: Server cannot response due to overloading or maintenance. The client can try again later
- **504 Gateway Timeout**: Proxy or Gateway indicates that it receives a timeout from an upstream server

(2) HTTP – Response: Response Header

- The **header** allows the server to pass additional information about the response that cannot be placed in the status line
- Header fields give information about the server and how to access the resource identified by the request URI
- The response headers are in the form of **name:value** pairs

field_name: field_value

Ex:

```
Content-Type: text/html
Content-Length: 35
Connection: keep-alive
Keep-Alive: timeout=15, max=100
```

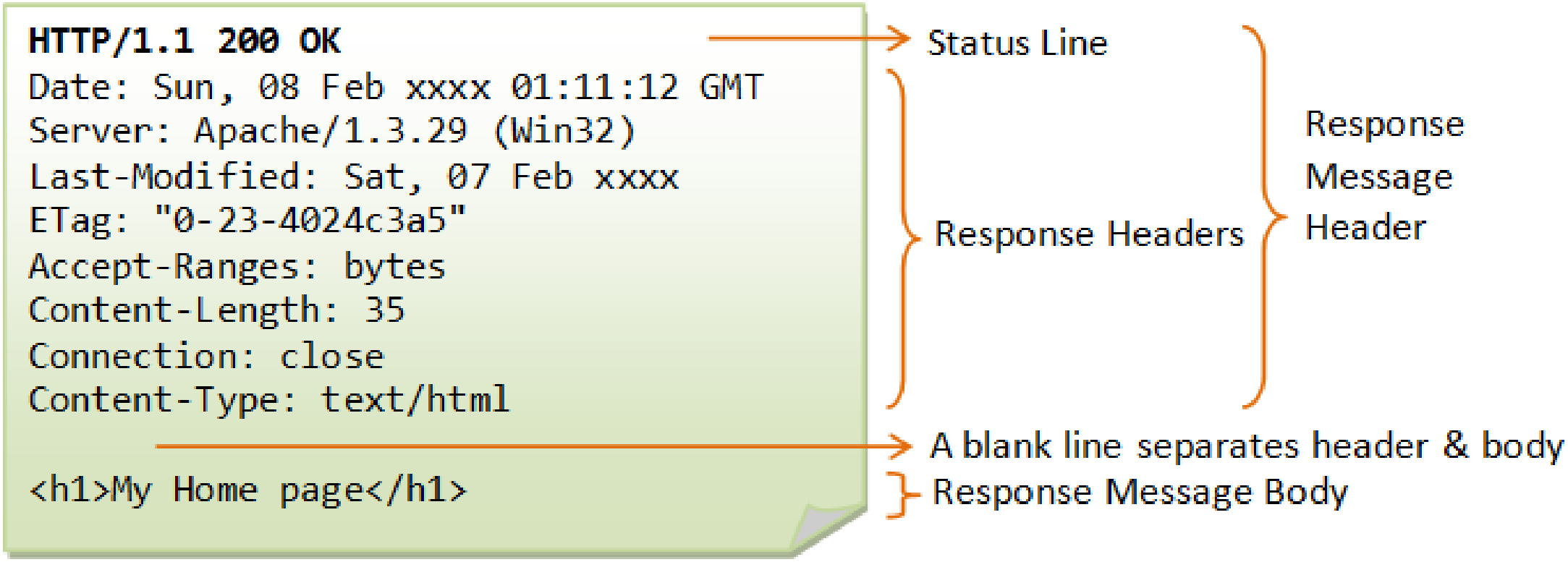
HTTP – Response: Response Header

- **Accept-Ranges** – Allows the server to indicate its acceptance of range requests for a resource
- **Age** – Estimate of the amount of time since the response was generated at the origin server
- **Location** – Redirects to a location other than the request URI for request completion or identification of a new resource
- **Proxy-Authenticate** – Allows the client to identify itself (or its user) to a proxy that requires authentication

(3) HTTP – Response: Message Body

- The **message body** in the response must also be preceded by a blank line
- The response to a HEAD request does not include a message body. All other responses do include a message body, although it may be of zero length
- The requested resource, e.g., the actual HTML, is included in the message body of the response

A Sample HTTP Response Message



Outline

- How the Internet Works: An Overview
  - Networking Concepts
  - The Internet
  - Internet Hot Topics

- **The HTTP Protocol**
  - HTTP Overview
  - HTTP Request
  - HTTP Response
  - **HTTP Sessions and Cookies**

HTTP Sessions

Session is used to refer to the time that a user first starts using a web application to the time they finished using that application

How Web Sessions Work:

1) The HTTP client (e.g., a browser) establishes a TCP connection with a server (typically port 80), and initiates a request

2) The HTTP server, listening on the port, receives the request, process it, and sends back a response. The response includes a **session ID**, and the server may store user information related to this session

3) When the browser makes another request, and includes the session ID, the server can reference previously stored information and respond accordingly

Cookies

- The **session ID** is typically a long randomly generated string sent back to the browser as a **cookie**
- The cookie is stored in the browser, and sent back to the server with every request
- Additional data, related to the session itself, can be stored in the cookie
- If not managed properly, this information can lead to security vulnerabilities

Sessions and Cookies

- It’s common in web applications to provide the session ID in the cookie, but to keep other information in a session store on the server side
- A cookie can only store about 4Kb of data – sometimes this is not enough
- Secret information should not be passes through cookies
- The session store uses the session ID to keep track of data