Ch02: Application Layer

Group Exercise
Desktop/Mobile Applications

Video #1 (09 minutes 10 seconds)

Time	Description
00:00	First node at UCLA
03:00	TCP/IP

Mobile phones make the Internet more accessible and the Internet makes mobile phones more useful

Vint Cerf

(the father of the Internet)

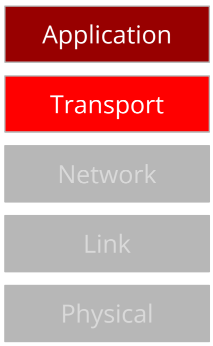
Video #2 (49 minutes)

Time	Description
00:00	First node at UCLA
04:00	Use of ARPANet for military (US DoD)
05:00	Consolidation of telephone line network with satellite & radio network
10:00	Tim-Berners Lee World-Wide-Web (HTTP), Mosaic Browser, Netscape Communication
14:00	The birth of the FIRST mobile phone technology (Motorola 1983)
15:20	Confluence of The Internet & Mobile Phone

Internet Layers

Application	*Exchange messages between two applications*
Transport	*Data transfer between two processes*
Network	*Data transfer between two hosts*
Link	*Data (frame) transfer between two neighboring network elements*
Physical	*Bit transfer on physical medium*

Layered Structure



Key benefit of the layered structure

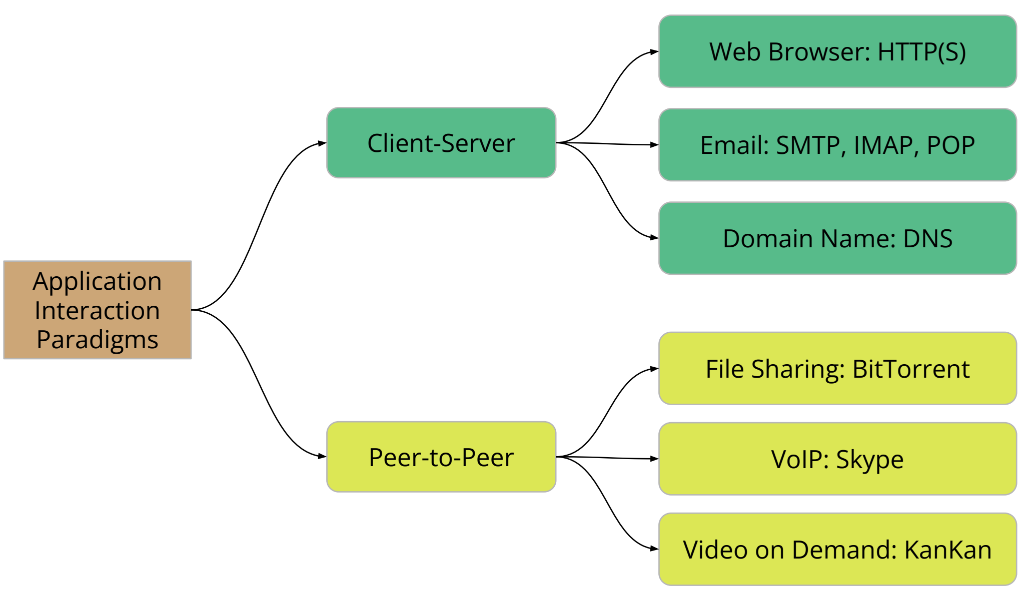
Your application needs to know only the services provided by the ***Transport Layer***

Chapter 2 Objectives

- Investigate various **application level protocols**
- Two types of application interactions
  - client-server: Web, Email, Domain Name
  - peer-to-peer: BitTorrent
- Applications & Protocol
  - Web: HTTP
  - Email: SMTP, IMAP
  - Domain Name: DNS
  - BitTorrent: Peer-to-Peer File Transfer

A protocol defines:

- Message syntax & semantic
- Order of messages (sent/received)
- Actions of the sender (*recipient*) upon sending (*receiving*) a message



Two interaction styles

Client-Server

- Server
  - Always-On Host
  - Have permanent IP address
  - Process that waits to be contacted
- Client
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with other client, only with server
  - Process that initiates communication

Peer-to-Peer

- No always-on server
- Peers request service from other peers
- Peers provide service in return to other peers
- Peers are intermittently connected and may use different IP addresses each they run

Socket

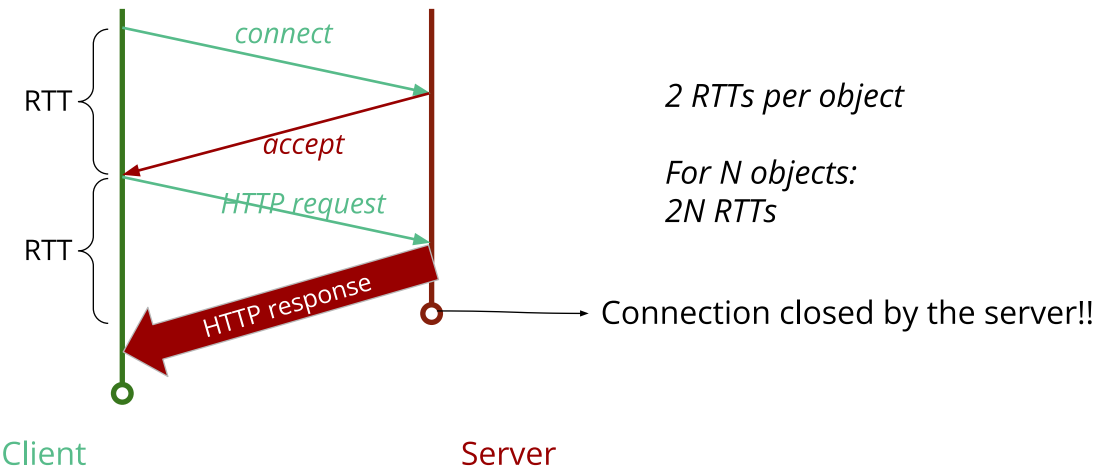
- Each socket is uniquely identified using a pair (IP addr & Port number)
  - **IP address** is associated with the (physical location) of the host
  - **Port** number is associated with which **process** within the host
- Recall that in the layered architecture
  - the **application layer** needs to know only how to interact with the **transport layer**
  - the transport layer is responsible for delivering data from one process to another process
- Use the socket library to interact with the transport layer
- Two services provided by the transport layer
  - TCP service: SOCK_STREAM
  - UDP service: SOCK_DGRAM

HyperText Transfer Protocol
HTTP

HTTP Versions

- HTTP 0.9: One line message (without header lines)
- HTTP 1.0: non-persistent connections
- HTTP 1.1
  - persistent connections (TCP connection stays open after a response)
  - chunked responses
  - client may send multiple asynchronous requests, but the server must respond in the order requested
- HTTP 2.0:
  - on multiple requests, the client can specify the response order to be returned by server
  - the server may push unrequested objects to client
- HTTP 3.0: security & congestion control

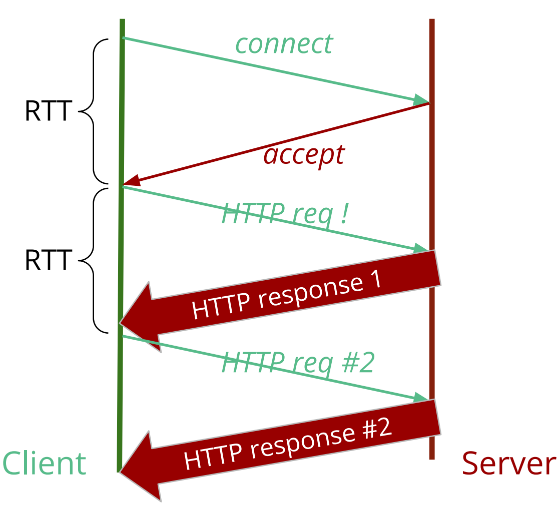
HTTP 1.0 Non-Persistent Connection



2 RTTs per object

For N objects:
2N RTTs

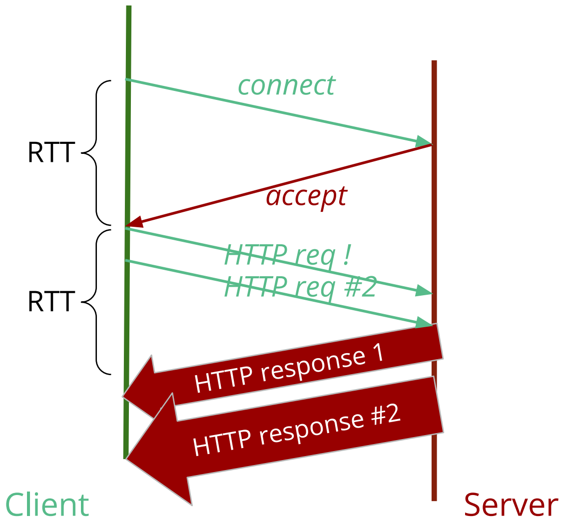
HTTP 1.1 Persistent Connection



1 RTT for initial connection
1 RTT for each object

For N objects:
N + 1 RTTs

HTTP 1.1 Persistent Connection & Pipeline



1 RTT for initial connection
1 RTT for each object

For N objects:
2 RTTs + N transfer times

Browser Demo
Developer Tools ⇒ Network Tab

HTTP Cookies

- Why need cookies?
  - HTTP itself is **stateless**
  - But sometimes both the web server and the browser need to maintain some state between transactions
- Implementation requires coordination between the web server and the browser
  - The web server initiates by sending a cookie in its response
  - The client (your web browser) saves the cookie in its local storage and uses it in its subsequent requests
  - Upon receiving the (same) cookie the server knows that the request(s) originate from the **same browser** (potentially the same host and the same user)

Practical Use of HTTP Cookies

- RFC2109
  - Typical size limit of HTTP Cookies is 4 kilo bytes
  - A browser is expected to be able to accept 300 cookies
- What is it?
  - A small data generated by the website which is unique to you and your browser
- Practical use: good or bad?
  - the web server remember the history of your last visit
  - show a web content based on your past visits
  - Skip login authentication
  - Track your online activities
  - ...

Third Party Cookies

- You are visiting a website at http://abc.org
  - Document fetch from http://abc.org may include object fetched from http://xyz.com
- Your web client has to fetch objects from both `abc.org` and `xyz.com`
  - HTTP requests to http://xyz.com includes a header line "referred by" http://abc.org
  - Cookies set from http://abc.org is a "first party" cookie
  - Cookies set from http://xyz.com is a **third party cookie**

HTTP 1.1 vs. HTTP 2.0

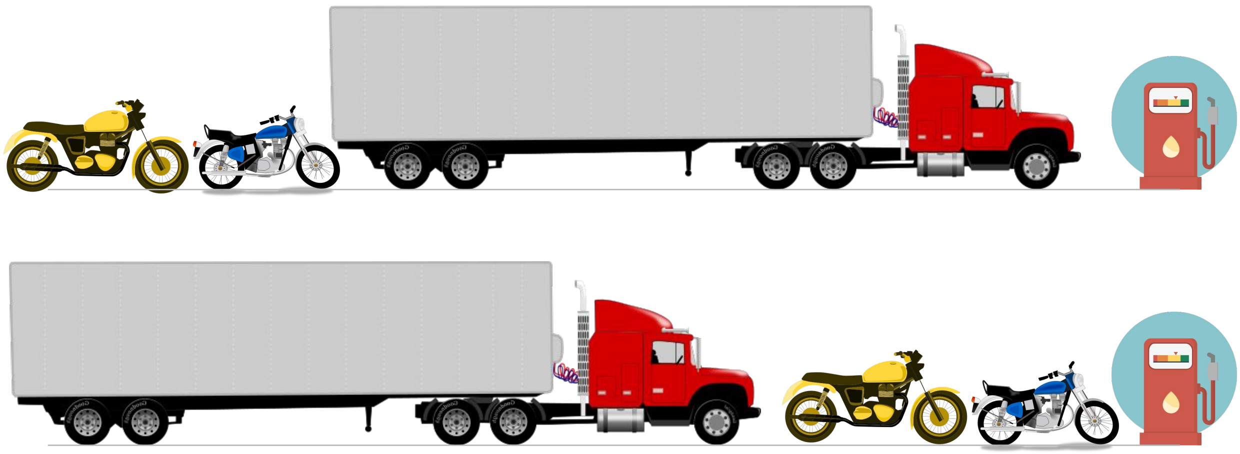
HTTP 1.1

- Multiple request in a pipeline
- Server responds in the order of the request

HTTP 2.0

- Multiple request in a pipeline
- Client may specified the priority of the object in the requests
- Server responds in the order of priority (which typically different from the order of the requests)
- Server may also push unrequested objects

Head of Line Blocking

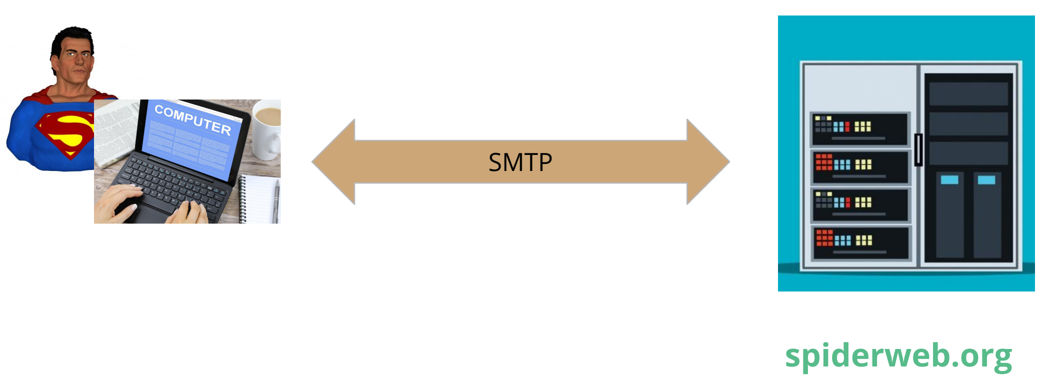


Simple Mail Transfer Protocol (SMTP)

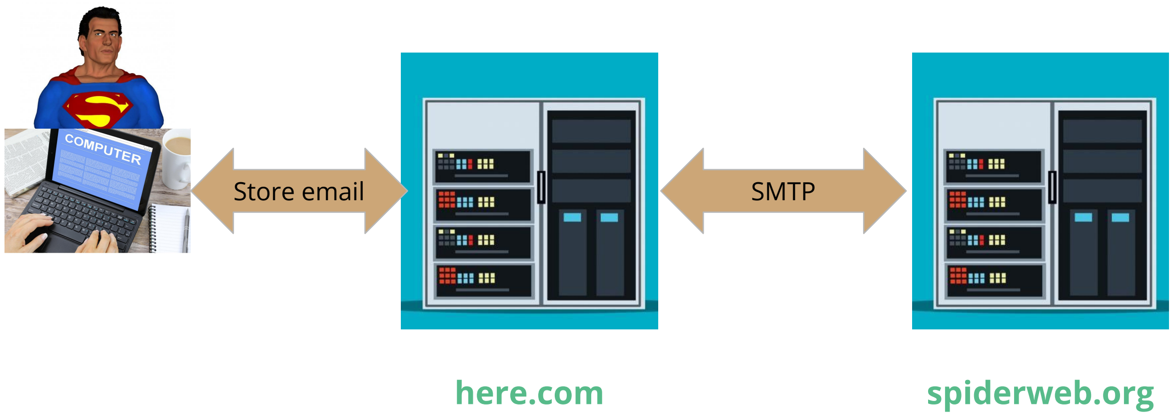
Composing & Sending emails

- Sender: superman@here.com
- Recipient: spiderman@spiderweb.org
- Components required (at the time of “me” sending the email)
  - An email client run by “superman” (on a laptop/smartphone/desktop)
  - Mail server at `spiderweb.org`
  - (Optional) Mailer program at `here.com`
    - Acting as a server w.r.t to “superman”’s email client
    - Acting as a client w.r.t to **spiderweb.org**
- SMTP protocol is used when *pushing emails* to the mail server at **spiderweb.org**

Sending emails option #1



Sending emails option #2



Retrieving & Reading emails

- Sender: superman@here.com
- Recipient: spiderman@spiderweb.org
- *Option 1 (less common today)*: Email client runs on the same machine as mail server spiderweb.org
    - The client program can directly open YOU's mailbox (a flat text file)
    - The mail server at **spiderweb.org** does not have to be running
- *Option 2*: Email client runs a desktop/laptop different from mail server there.org
    - The email client downloads the emails from mail server **spiderweb.org** using either IMAP, HTTP, or POP

Reading emails option #1 (less common today)

Reading emails option #2 (more common today)

Web Email Demo: GVSU GMail (Show Original Text Content)

Business/Developer Opportunity

- Incorporate Digital Signature (RSA or similar) to email client
- Recipient
- Prevent alteration of forwarded email
- Secure SMTP

Domain Name Server

DNS Messages

- RFC1035
- Additional Helpful Information
  - Mailbox Domain Name
  - Host Information (HINFO): CPU, OS (listed in RFC 1010)
  - MX Server Information (MINFO)
    - Server Name for Mailing List
    - Server Name for Receiving Mail Errors (“bounced back emails”)

Video Streaming

Requirements: steady stream of video frame rate (24 frames/second)

Format	Frame Size	Mega pixels/second*
Standard Definition	720 x 480 pixels	8.3
High Definition	1280 x 720 pixels	22
Full HD	1920 x 1080 pixels	50
2K	2560 x 1440 pixels	88.5
4K	3840 x 2160 pixels	200 Mbps
8K	7680 x 4320 pixels	800 Mbps

- *these numbers are raw calculations, video data are not encoded and uncompressed*
- *1 RGB pixel may require 24 bits (8-bit RED, 8-bit GREEN, 8-bit BLUE)*

Video Streaming

- Issues
  - Network delay may vary, video frames may arrive at the client side at *irregular intervals*
  - Available bandwidth is much lower than the required bandwidth (*even at 60% compression ratio*)
- Possible solutions
  - Use a buffer at the client side to accumulate incoming video frames
  - Delay the playback of the video until the buffer has “sufficient” number of rames
- Better solution
  - Encode the video to support multiple (transmission) bit rates
  - Let the client decide (at runtime) which bitrate it prefers
  - When less bandwidth is available, the client asks the server to switch to a lower bitrate (and vice versa)

DASH: Dynamic Adaptive Streaming over HTTP

- The entire video file is divided into multiple chunks (N chunks)
- Each chunk is encoded at several different bit rates (R choices)
- Store each N x R files separately
  - Separate files and/or separate server locations
  - Use a manifest file to map the URL of each file
- At playback time, the client
  - Estimate the available bandwidth (B_e)
  - Use HTTP to request one chunk at a time
  - When the available bandwidth changes, the client requests the next chunk encoded at a different rate (lower or higher)

CDN: Content Distribution Networks

- To avoid a single server bottleneck, (file) contents are replicated across multiple server (“*geographically” spaced out*)
- When a client request for a specific file, the (main) server responds with a list of (clone) servers where the file is available
  - A DASH manifest file when this technique is used for video streaming
- Alternative to a list of clones, the content distribution can be implemented by dynamically changing the DNS records (same host name but different IP address)
- The client then picks the best (clone) server to pull the actual file from