CSC 4350 – Computer Networks

Fall 2024 Semester

Lecture 5 – The Application Layer/http Protocol

Note

- Material in this lecture is heavily borrowed from Kurose & Ross' "Computer Networking: A Top Down Approach, 8th Edition"

Reliable Data Transfer

- If a protocol provides a guaranteed data delivery service
- One important service that a transport-layer protocol can provide is process-to-process reliable data transfer
- With this service, sending process can just pass its data into the socket and know with confidence that the data will arrive without errors at the receiving process
- Loss-tolerant applications
  - Transport-layer protocol doesn’t provide reliable data transfer, some data may not arrive at the receiver
  - Most notably – multimedia applications, such as A/V, conversational A/V that can tolerate a loss
  - Loss of data may result in a small glitch in A/V, but not a “crucial” impairment

Throughput

- Applications that have throughput requirements are called bandwidth-sensitive applications
  - Many current multimedia applications are bandwidth sensitive
  - Some multimedia applications may use adaptive coding techniques to encode digitized voice or video at a rate that matches the currently available throughput
- Elastic applications – specific throughput requirements
  - Can make use of as much or little throughput as is available
  - Examples
    - SMTP/Email
    - FTP
    - Web transfers

Timing

- Transport layer protocol can provide timing guarantees
  - Can be variable
- Example: guarantee that every bit the sender pumps into the socket arrives at the receiver's socket no more than 100 ms later
  - Appealing to interactive real-time applications
    - Internet telephony, virtual environments, conferencing
    - Each provide tight timing constraints on data delivery in order to be effective
- Non-real-time applications – lower delay is always preferable to higher delay, but no tight constraint is placed on end-to-end delays

Security

- Possibly one or more security systems
- Example
  - Sending host – a transport protocol can encrypt all data transmitted by the sending process
  - In the receiving host – transport layer protocol can decrypt the data before delivering the data to the receiving process
  - Provides confidentiality between two processes, even if data is observed in between sender/receiver
- Other ways to achieve confidentiality?

Figure 2.4 – Requirements of Selected Network Applications

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Smartphone messaging	No loss	Elastic	Yes and no

Transport Services Provided by the Internet

- TCP Services
  - Connection-oriented service
  - Reliable data transfer service
- UDP Services
- Services Not Provided by Internet Transport Protocols

TCP – Connection-Oriented Service

- Has client and server exchange transport-layer control information with each other before the application-level messages start moving
- Alerts the client and server, allowing them to prepare for incoming packets
- After this “handshaking phase,” a TCP connection is said to exist between the sockets of the two processes
  - Full-duplex connection in that the two processes can send messages to each other over the connection at the same time
  - When application finishes sending messages, it must tear down the connection

TCP – Reliable Data Transfer Service

- Communicating processes can rely on TCP to deliver all data sent without error and in the proper order
- When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing/duplicate bytes
- Provides a congestion-control mechanism
  - Throttles a sending process when the network is congested between sender and receiver
  - Attempts to limit each TCP connection to its fair share of network bandwidth

UDP Services

- Lightweight transport protocol, providing minimal services
- Connectionless – no handshaking before the two processes start to communication
- Unreliable data transfer service
  - No guarantee that message sent will ever reach the receiving process
  - Messages may arrive out of order
- No congestion-control mechanism – sending side of UDP can send however much data into the network layer at any rate

Services Not Provided by Internet Transport Protocols

- What about throughput or timing guarantees?
  - Services not provided by today's protocols
- Applications work well-ish because they have been designed to cope with the lack of throughput/timing guarantees
- Figure 2.5 – transport protocols utilized by some popular Internet applications
- TCP gets chosen a bit – reliable data transfer, guaranteeing all data will eventually get to the destination

Figure 2.5 – Popular Internet Applications, Application Layer Protocols, Transport Protocols

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP 1.1 [RFC 7230]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube), DASH	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

Application-Layer Protocols

- Defines how an application's process, running on different end systems, pass messages to each other
- Particulars
  - Types of messages exchanged; request messages and response messages, for instance
  - Syntax of the various message types, such as fields in the message and how fields are delineated
  - Meaning of the information in the fields
  - Rules for determining when and how a process sends/responds to messages
- Some application-layer protocols are specified in Requests for Comments (RFCs)
  - HTTP – RFC 7230
  - If a browser follows the rules of the HTTP RFC, the browser will be able to retrieve web pages from any web server that has also followed the HTTP RFC
  - Other application-layer protocols are proprietary and not available in the public domain
    - Skype
- A protocol is one piece of a network application

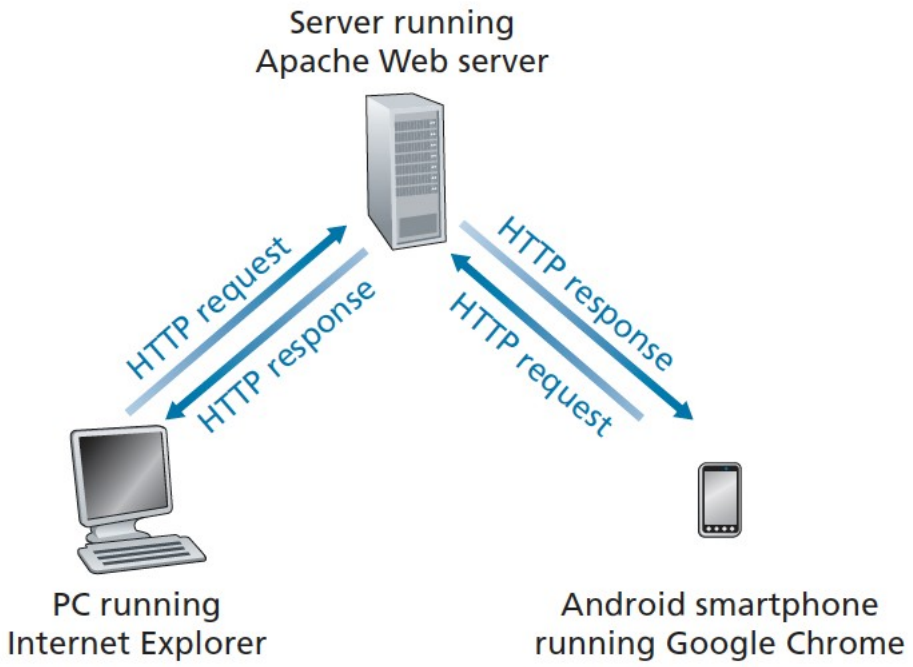
The Web and HTTP

- Overview of HTTP
- Non-Persistent and Persistent Connections
- HTTP Message Formats
- User-Server Interaction – Cookies
- Web Caching
- HTTP/2

Overview of HTTP

- HTTP (HyperText Transfer Protocol) – Web's application-layer protocol – heart of the Internet
- Defined in RFCs 1945, 7230, and 7540
- Implemented with two programs
  - Client
  - Server
  - Interact between each other by exchanging messages
- Web page – consists of objects, usually HTML file, image, CSS, etc.
- Clients – web browsers
- Servers – responsible for handling requests/sending data back to clients

Figure 2.6 – HTTP Request-Response Behavior



Overview of HTTP

- Uses TCP as its transport protocol
  - Client first initiates a TCP connection with the server
  - Once established, the browser and server processes access TCP through socket interfaces
  - Socket
    - Client-side – “door” between client and connection
    - Server-side – “door” between server process and connection
  - Client sends HTTP request messages into its socket interface and receives HTTP messages from its socket interface
  - Server receives requests from socket and sends response messages via socket
  - Once message is sent by client, it is out of client’s hands, and instead with TCP
  - Implication – each HTTP request message sent by a client will eventually reach the server; each server response will eventually be received by the client
- Server sends requested files without storing any state information on the client
  - If client asks for the same object/piece – server will send it again

Non-Persistent and Persistent Connections

- Application developer needs to make a decision wrt to messages going back and forth
  - Should each request/response pair be sent over a separate TCP connection or sent over the same TCP connection
    - Non-persistent connections
    - Persistent connections

HTTP With Non-Persistent Connections

- Steps of transferring a web page from server to client
- Page consists of base HTML file and 10 JPEG images; all 11 objects reside on the same server
- URL
  - http://www.someSchool.edu/someDepartment/home.index

Non-Persistent – Steps

- The HTTP client process initiates a TCP connection to the server www.someSchool.edu on port 80 (default port for HTTP)
  - Associated with the TCP connection, there will be a socket at the client and a socket at the server
- The HTTP client sends an HTTP request message to the server via its socket; request message includes the path /someDepartment/home.index
- The HTTP server process receives the request message via socket and retrieves the object /someDepartment/home.index from its storage
  - Encapsulates the object in an HTTP response message
  - Sends the response message to the client via its socket

Non-Persistent – Steps

- HTTP server process tells TCP to close the TCP connection
  - TCP doesn’t actually terminate the connection until it knows for sure that the client has received the message intact
- HTTP client receives the response message
  - TCP connection terminates
  - Message indicates that the encapsulated object is an HTML file
  - Client extracts the file from response message, examines HTML file and finds references to 10 JPEG objects
  - First four steps repeated for each of the referenced JPEG objects

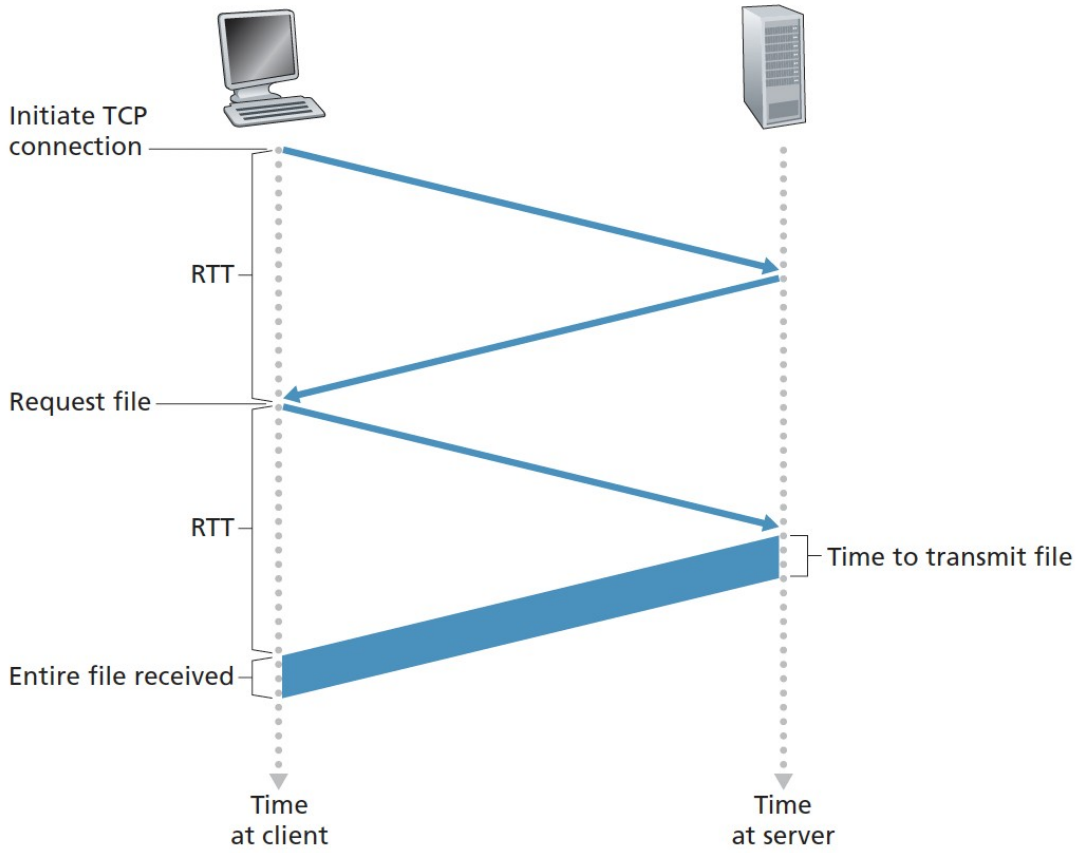
HTTP – Non-Persistent Connections

- Two browsers may interpret a web page in somewhat different ways (not as bad as it used to be)
- HTTP has nothing to do with how a page is interpreted by the client; specifications define only protocol between client and server
- Users can configure browsers to control the degree of parallelism
  - Browsers may open multiple TCP connections and request different parts of the web page over multiple connections

“Back of Envelope” Calculation – Elapsed Time

- Define round-trip time (RTT)
  - Time it takes for a small packet to travel from client to server and then back to the client
  - Includes packet-propagation delays, packet-queuing delays in intermediate routers and switches, and packet-processing delays
  - Shown in Figure 2.7 (next slide)
  - Browser clicks on link, generates TCP connection with server
    - Involves a “three-way” handshake” – client sends a small TCP segment to the server
    - Server acknowledges and responds back with a small TCP segment
    - Client acknowledges back to the server
    - First two parts of the three-way handshake take one RTT
    - Then – client sends HTTP request message combined with the third part of the three-way handshake (the acknowledgment)
    - Once request arrives at the server, server sends the HTML file into the TCP connection
      - Requires another RTT
  - Response time: 2RTT + transmission time at the server of the HTML file

Figure 2.7 – Back-of-the-Envelope Calculation for Time to Request/Receive HTML File



HTTP with Persistent Connections

- Non-persistent connections – issues
  - A brand-new connection must be established and maintained for each requested object
  - For each connection, TCP buffers must be allocated and TCP variables must be kept in both client and server
    - Significant burden – if server handling several requests from different clients all at the same time
  - Also: each object suffers a delivery delay of 2RTT
- HTTP/1.1 persistent connections – server leaves the TCP connection open after sending a response
  - Subsequent requests and responses between the same client and server can be sent over the same connection
  - Multiple web pages residing on the same server can be sent from server to client over a single persistent TCP connection
  - Requests for objects can be made back-to-back without waiting for replies to pending requests (pipelining)

HTTP Message Format – Message

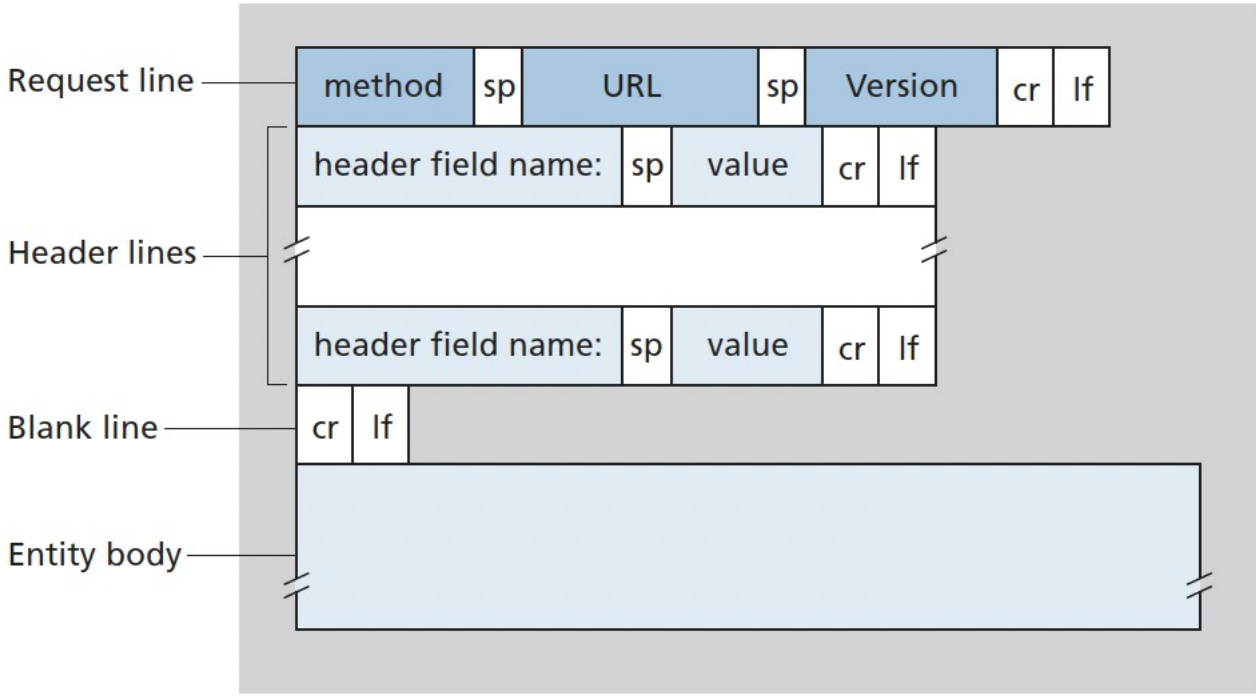
- Message is written in ordinary ASCII – computer-literate human can read it
- Message consists of five lines, each with carriage return and a line feed
- Last line is followed by an additional carriage return and line feed

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

HTTP Message Format – Message

- First line of an HTTP request is the request line
  - Three fields
    - Method; possible values: GET, POST, HEAD, PUT, DELETE
    - URL field
    - HTTP version field
- Remaining lines: header lines
  - Host: specifies the host on which the object resides
  - Connection: the browser is telling the server that it doesn’t want persistent connections
  - User-agent: browser type making the request (Mozilla/5.0)
  - Accept-language: user prefers to receive a French version of the object, if such an object exists on the server; otherwise it sends its default version
- Majority of HTTP request messages use the GET method
  - Retrieve an object

Figure 2.8 – General Format of an HTTP Request Message



Message Format

- General format closely follows the earlier example
- After header lines, there is an “entire body.”
  - Empty with the GET method
  - Used with the POST method – when the user fills out a form
- The HEAD method is similar to GET
  - When server receives a request, it responds with an HTTP message but leaves out the requested object
  - Developers use HEAD for debugging
- PUT – often used with web publishing tools
  - Allows a user to upload an object to a specific path/directory on a specific web server
  - Also used by applications that need to upload objects to web servers
- DELETE – allows user/application to delete an object on a web server

HTTP Response Message

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

- Three sections
  - An initial status line
    - Three fields
      - Protocol version
      - Status code
      - Corresponding status message
      - Example: server is using HTTP/1.1 and that everything is okay (object is being sent)
  - Six header lines
    - Connection: close – tell the client it is going to close the TCP connection after sending the message
    - Date: indicates date and time when HTTP response was created
    - Server – indicates the message was generated by an Apache Web server (similar to User-agent in message request)
    - Last modified – time and date when the object was created/last modified
      - Critical for caching
  - Entity body – the meat of the message – the requested object

Figure 2.9 – General Format of an HTTP Response Message