Contents

Application Layer Protocols

- Network Application Architectures
- WWW and HTTP
- Simple Mail Transfer System (SMTP)
- Domain Name System (DNS)
- P2P applications
- Web applications

Connecting a Network App

- 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 allow for rapid app development propagation
- Possible structure of applications
  - Client-Server
  - Peer-to-Peer (P2P)

Client-Server Architecture

- Server
  - Always-on host
  - Permanent IP address
  - Data centres for scaling
- Clients

- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

P2P Architecture (e.g. Bittorrent)

- *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

Processes

- Process: program running within a host
- Within same host, two processes communicate using *inter-process communication* (defined by OS)
- Processes in different hosts communicate by exchanging *messages*
- Applications with P2P architectures have client processes and server processes

Client process: process that initiates communication

Server process: process that waits to be contacted

What Transport Services Does an App Need?

Data Loss

- Some apps (e.g. file transfer, web transactions) require 100% reliable data transfer
- Other apps (e.g. audio) can tolerate some loss

Timing

- Some apps (e.g. Internet telephony, interactive games) require low delay to be "effective"

Throughput

- Some apps (e.g. multimedia) require minimum amount of throughput to be "effective"
- Other apps ("elastic apps") make use of whatever throughput they get

Security

- Encryption, authentication, data integrity, …

Application	Data Loss	Throughput	Time Sensitive
File transfer	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic	No
Real-time	Loss-tolerant	Audio: 5kbps-10Mbps; Video: 10kbps-5Mbps	Yes, 100 msec
Stored audio/video	Loss-tolerant	Same as above	Yes, few secs
Interactive games	Loss-tolerant	Few kbps up	Yes, 100 msec
Text messaging	No loss	Elastic	Yes and no

WWW and HTTP

- A web page consists of objects
  - An object file can be HTML file, JPEG image, Java applet, audio file, …
- A web page consists of base HTML file which includes several referenced objects
  - Each object is addressable by a uniform resource locator (URL)

`www.someschool.edu/someDept/pic.gif`

- `www.someschool.edu`: host name
- `/someDept/pic.gif`: path name

Hyper Text Transfer Protocol (HTTP)

- HTTP uses TCP
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP msgs exchanged between Web browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- HTTP is “stateless”
  - Server maintains no information about past client requests
  - More efficient

HTTP Connections

- Non-persistent HTTP
  - At most one object sent over TCP connection
  - Connection then closed
  - Downloading multiple objects requires multiple connections
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection between client and server
- Default mode of HTTP
  - Persistent connections with pipelining

Non-Persistent HTTP

Suppose user enters URL: `www.someSchool.edu/someDept/home.index`

1. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80
2. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. “accepts” connection, notifying client
3. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDept/home.index`

4. HTTP server receivers request message, froms *response message* containing requested objects, and sends message into its socket. HTTP server closes TCP connection
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects.

Steps repeated for each jpeg object.

Response time:

- One RTT initiate TCP connect
- One RTT for HTTP request, and first few bytes of HTTP response to return
- File transmission time
- Non-persistent HTTP response time
  - 2RTT + file transmission time

Persistent HTTP

Non persistent HTTP issues:

- Requires 2 RTTs per object
- OS overhead for each TCP connection
- Browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server sent over open connection
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

HTTP Request Message

- Two types of HTTP messages
  - *request*, *response*
- HTTP request message
  - ASCII (human-readable format)

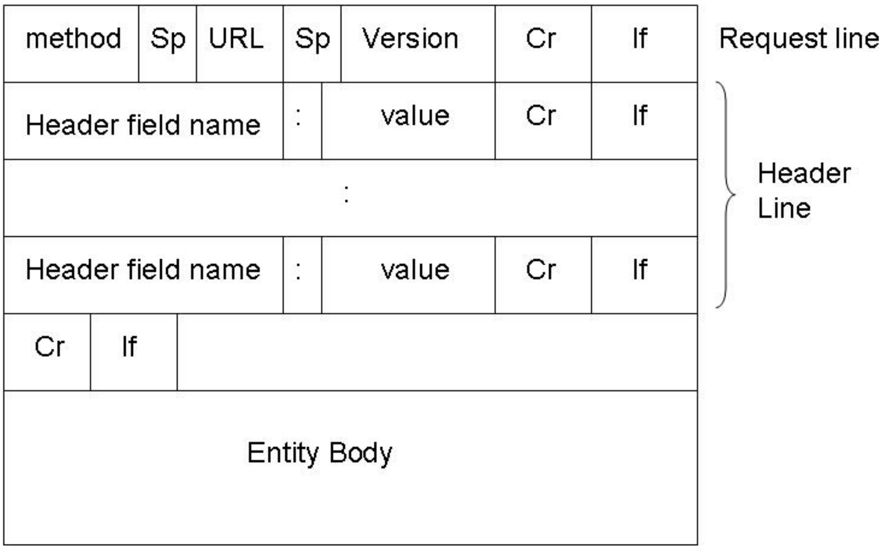

Figure 1: Request Message

Method Types

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
    * Used in conjunction with Web publishing tools
- DELETE
  - Deletes file specified in the URL field

Uploading Form Input:

- POST Method
  - Web page object includes form input
  - Input is uploaded to server in entity body
  - Used when sending data to the server changes its state
- URL Method
  - Uses GET method
  - Input is uploaded in URL field of request line

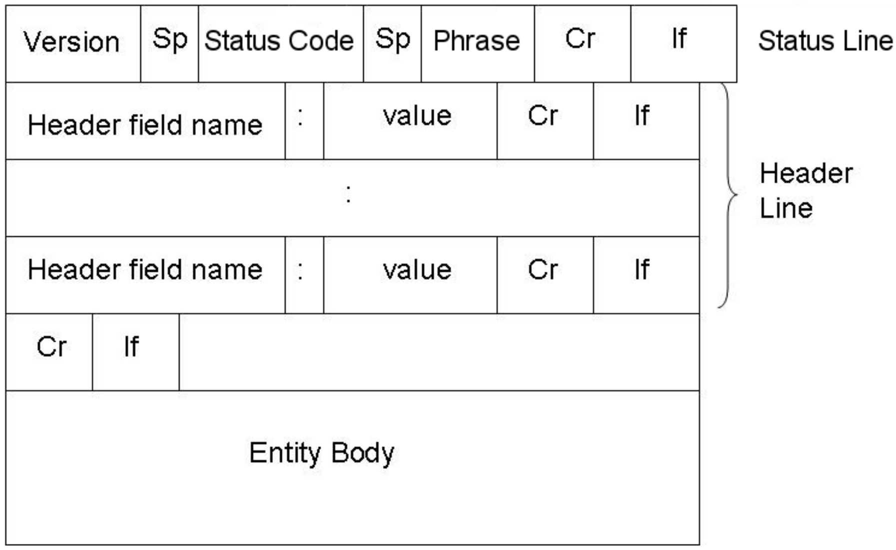

Figure 2: Response Message

Response Codes

- Status code appears in first line in server-to-client response message
- Some sample codes
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request
  - 404 Not Found
  - 505 HTTP Version Not Supported

Cookies

- What can cookies be used for?
  - Authorisation
  - Shopping carts
  - Recommendations
  - User session state (Web mail)
- How to keep "state"
  - HTTP has been designed as a stateless protocol
  - Maintain state at sender/receiver over multiple transactions
    * Cookies: HTTP messages carry state

Privacy

- Cookies permit sites to learn a lot about you
- You may supply name and e-mail to sites

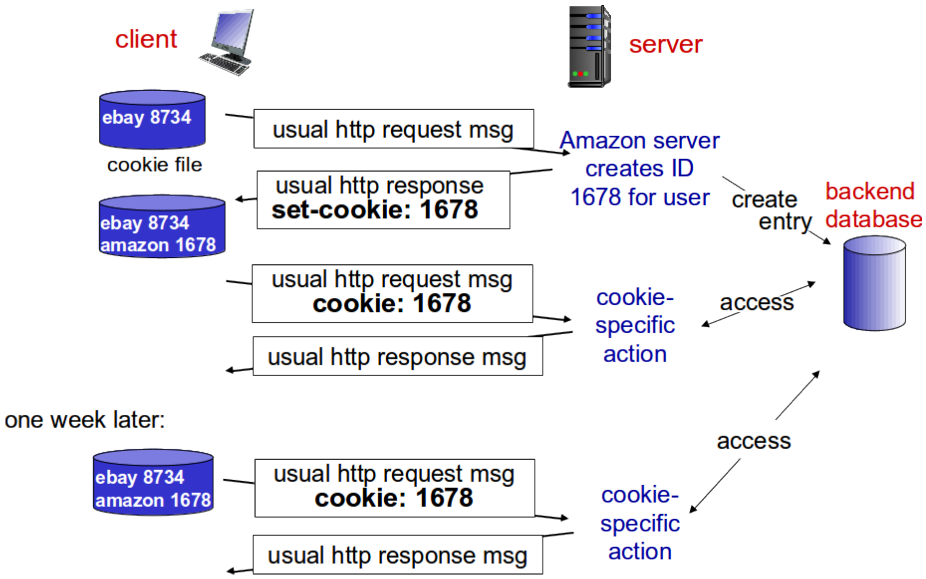

Figure 3: Keeping State

Web Caches (Proxy Server)

Goal: Satisfy client request without involving origin server

- User sets browser to access the web via proxy
- Browser sends all HTTP requests to proxy server
  - Object in cache
    - Proxy returns object
  - Else proxy requests object from origin server
    - Returns object to client
- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache is installed by ISP

  - University, company, residential ISP
- Why web caching?
  - Reduce response time for client request

More About Web Caching

- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache is installed by ISP
  - University, company, residential ISP

Q: Why Web caching?

- Reduce response time for client request
- Reduce traffic on an institution's access link

Caching Example

Fatter Access Link

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers: 15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from Institutional router to any origin server: 2 sec (Internet Delay)
  - Access link rate: 154 Mbps
- Consequences
  - LAN utilisation: $15 \times \frac{10^5}{10^9} = 0.15\%$
  - Access link utilisation: 0.97%
    - $15 \times \frac{10^5}{1.54} \times 10^6$
  - Total delay = Internet Delay + Access Delay + LAN Delay
    - $= 2$ sec

Install Local Cache

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers: 15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from Institutional router to any origin server: 2 sec (Internet Delay)
  - Access link rate: 1.54 Mbps
- Consequences
  - LAN utilisation: 0.15%
  - Access link utilisation= ?
  - Total delay = ?
- Cost: Web cache (cheap!)

Conditional GET

Goal: Do not send object if cache has up-to-date cached version (no object transmission delay)

- Cache: specify date of cached copy in HTTP request
  - If-modified-since:
- Server: response contains no object is cached copy is up-to-date
  - HTTP/1.0.304 Not Modified

Electronic Mail

- Three major components
  - User Agents
  - Mail Servers
  - Simple Mail Transfer Protocol (SMTP)
- User Agent
  - a.k.a. “mail client”
  - Composing, editing, reading mail messages
  - e.g. Outlook, Thunderbird, iPhone mail client
  - Outgoing, incoming messages stored on server

Mail Servers

- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - Client: sending mail server
  - Server: receiving mail server

SMTP

- Uses TCP to reliably transfer email message from client to server on port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - Handshake (greeting)
  - Transfer of messages
  - Closure
- Command/response interaction (like HTTP, FTP)
  - Commands: ASCII text
  - Response: status code and phrase

Alice Sends Message to Bob

1. Alice uses UA to compose message "to" bob@someschool.edu
2. Alice's UA sends message to her mail server
   - Message placed in queue
3. Client side of SMTP opens TCP connection with Bob's mail server
4. SMTP client sends Alice's message over the TCP connection
5. Bob's mail server places the message in Bob's mailbox
6. Bob invokes his user agent to read message

Mail Message Format

- RFC 822: standard for text message format
  - Header lines
    * To
    * From
    * Subject

  - Body
    * The message
    * ASCII characters only

Final Words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF to determine end of message

Comparison with HTTP

- HTTP: pull
- SMTP: push
- Both have ASCII command/response interactions, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects in a single message

Multipurpose Internet Mail Extension (MIME)

- Can only send messages in 7-bit ASCII format
  - Cannot be used for other languages e.g. French, Chinese, etc.
- MIME is a supplementary protocol that allows non-ASCII data to be send via SMTP
  - Converts non-ASCII to ASCII and vice versa

  Don't need to understand a huge amount about Header

Mail Access Protocols

- SMTP
  - Delivery/storage to receiver's server
- Mail Access Protocol
  - Retrieval from server
- POP: Post Office Protocol (RFC 1939)
  - Authorisation, Download
- IMAP: Internet Mail Access Protocol (RFC 1730)
  - More features, including manipulation of stored messages on server
- HTTP
  - Gmail, Hotmail, Yahoo! mail, etc.

POP3 Protocol

Authorisation Phase

- Client commands
  - user: declare username
  - pass: password
- Server responses
  - +OK
  - -ERR

Transaction phase, client:

- list: list message by numbers
- retr: retrieve message by number
- dele: delete
- quit

POP3 and IMAP

POP3

- "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- "download and keep"
  - Copies of messages on different clients
- Stateless across sessions

IMAP

- Keeps all messages in one place: at server
- Allows user to organise messages in folders
  - Names of folders between message IDs and folder names
- Keeps user states across sessions

Domain Name System (DNS)

- Distributed database implemented in hierarchy of many name servers
- Application-layer protocol
  - Hosts, name servers communicate to resolve names (address/name translation)
- DNS services
  - Hostname to IP address translation, e.g. `www.rte.ie` or `rte.ie`
  - Host aliasing
  - Canonical, alias name
  - Mail server aliasing
  - Load distribution
  - Replicated web servers
- Not one-to-one names to IPs/servers
  - Many IP addresses correspond to one name (redundant services)
  - Many names correspond to one IP address (virtual hosts)

A Distributed Hierarchical Database

- Client wants IP for `www.amazon.com`
- Client queries *root* server to find `.com` DNS server
- Client queries `.com` DNS server to get `amazon.com` DNS server
- Client queries `amazon.com` DNS server to get IP address for `www.amazon.com`

Root Name Servers

- Contracted by local name server that can not resolve name
- Root name server
  - Return list of IP addr for responsible TLD servers
  - `www.digwebinterface.com`

TLD and Authoritative Servers

- Top-level Domain (TLD) Servers
  - Responsible for `com`, `org`, `net`, `edu`, `aero`, `jobs`, `museums`, and all top-level country domains, e.g. `uk`, `fr`, `ca`, `jp`
    * Network Solutions maintains servers for `.com` TLD
    * Educause for `.edu` TLD

- Authoritative DNS Servers
  - Organisations' own DNS servers, providing authoritative hostname to IP mappings for organisations' named hosts

Local DNS

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - Also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - Has local cache of recent name-to-address translation pairs (but may be out of date)
  - Acts as proxy, forwards query into hierarchy

Name Resolution and Caching

- Recursive query
  - Puts burden of name resolution on contracted name server
- Once a name server learns of a mapping, it caches it
  - Cache entries timeout after some time (TTL)
  - TLD servers typically cached in local name servers
  - Root name servers not often visited!

Caching and Updating Records

- Once (any) name servers learns mapping, it caches mapping
  - Cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- Cached entries may be out-of-date (best effort name-to-address translation!)
  - If name host changes IP address, may not be known Internet-wide until all TTLs expire

DNS Records

- Distributed database storing resource records (RR)

RR format: `(Name, Class, Type, TTL, Value)`

- A: IPv4 address record
- AAAA: IPv6 address record
  - Where to go next
- NS: Name Server Record
- CNAME: Canonical Name (i.e. alias) record
- MX: Mail Exchange record

A & AAAA Records

- A Record - IPv4 address record

`www.ripe.net. IN A 193.0.6.139`

- AAAA Record - IPv6 record

'www.ripe.net. IN AAAA 2001:67c:2e8:22::c100:68b

NS Record

- Name is domain (e.g. foo.com)
- Value is hostname of authoritative server for this domain

NAME	TTL	TYPE	DATA
ns.example.com.	1800	A	192.168.1.2
example.com.	1800	NS	ns.example.com

CNAME Record

- Name is alias for some canonical (real) name

NAME	TTL	TYPE	DATA
www.example.com.	1800	A	192.168.1.2
ftp.example.com.	1800	CNAME	www.example.com

MX Record

- Value is name of mail server associated with name

NAME	TTL	TYPE	DATA	MX LEVEL
mail1.example.com.	1800	A	192.168.1.2	10
mail2.example.com.	1800	A	192.169.1.4	20
example.com.	1800	MX	mail1.example.com.	10
example.com.	1800	MX	mail2.example.com.	20
example.com.	1800	MX	mail100.backupexample.com.	100

DNS Protocol, Messages

- *query* and *reply* messages, both with same message format

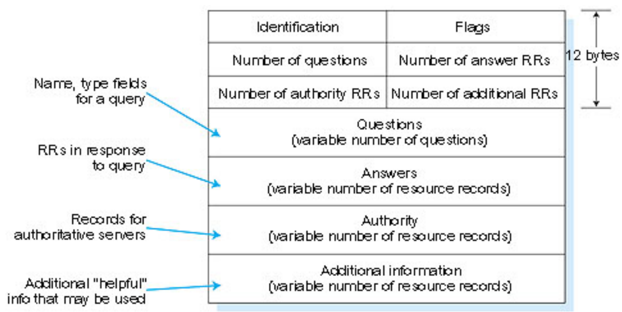

Figure 4: DNS Message Format

- Identification
  - 16 bit number for query
  - Reply to query uses same number

- Flags
  - Query or reply
  - Recursion desired
  - Recursion available
  - Reply is authoritative
- Questions
  - Name, type fields for query
- Answers
  - RRs in repsonse to query
- Authority
  - Records for authoritative servers
- Additional Info
  - Additional "helpful" info that may be used

Inserting Records into DNS

- Example: new startup "Network Utopia"
- Register name `networkutopia.com` at DNS registrar (e.g. Network Solutions)
  - Provide names, IP address of authoritative name server (primary and secondary)
  - Registrar inserts two RRs into `.com` TLD server
    - `(networkutopia.com, dns1.networkutopia.com, NS)`
    - `(dns1.networkutopia.com, 212.212.212.1, A)`
- Create
  - Authoritative server type A record for `www.networkutopia.com`
  - Type MX record for `networkutopia.com`

Attacking DNS

DDoS Attacks

- Bombard root servers with traffic
  - Not successful up to date
  - Local DNS servsers cache IPs of TLD servers, allowing root server bypass

Redirect Attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS server, which caches

Pure P2P Architecture

- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses
- Examples
  - File distribution (BitTorrent)
  - Streaming (KanKan)
  - VoIP (Skype)

File Distribution

- Question: how much time to distribute file (size F) from one server to N peers?
  - Peer upload/download capacity a limited resource

Client-Server

- Server transmission: must sequentially send (upload) N file copies
  - Time to send one copy: F/u_s
  - Time to send N copy: NF/u_s
- Client: each client must download file copy
  - d_{min} = min client download rate
  - min client download rate/max client download time: F/d_{min}

P2P

- Server Transmission: must upload at least one copy
  - Time to send one copy: F/u_s

- Client: each client must download file copy
  - Min client download time: F/d_{min}
- Clients: as aggregate must download NF bits
  - Max upload rate is $u_s + \sum u_i$

BitTorrent

- File divided into 256KB chunks
- tracker: tracks peers participating in torrent
- torrent: group of peers exchanging chunks of a file
- Peers in torrent send/receive file chunks
- Peer joining torrent
  - Has no chunks, but will accumulate them over time from other peers
  - Registers with *tracker* to get list of peers, connects to subset of peers ("neighbours")
- While downloading, peers uploads chunks to other peers
- Peer may change peers with whom it exchanges chunks
- Churn: peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

Requesting chunks

- At any given time, different peers have different subsets of file chunks
- Periodically, Alice asks each peer for a list of chunks that they have
- Alice requests missing chunks from peers
  - Rarest first

Sending chunks

- Alice sends chunks to those four peers currently sending her chunks at highest rate
  - Other peers are choked by Alice
  - Re-evaluate top 4 every 10 secs
- Every 30 secs: randomly select another peers, starts sending chunks
  - "Optimistically unchoke" this peer
    * Newly chosen peer may join top 4

Tit-for-tat

- Alice “optimistically unchokes” Bob
  - Alice becomes one of Bob’s top-four provides
- Bob reciprocates
  - Bob becomes one of Alice’s top four providers

Web Applications

- Usability of web applications has lagged behind that of desktop applications
- Rich Internet Applications (RIAs)
  - Web applications that approximate the look, feel and usability of desktop applications
- Two key attributes
  - Performance and rich GUI
- RIA performance
  - Comes from Ajax which uses client-side scripting to make web applications more responsive

Synchronous Web Communication

- Synchronous: user must wait while new pages load
  - Typical communication pattern used in web pages (click, wait, refresh)
- While a synchronous request is being processed on the server, the user cannot interact with the client web browser

Web Applications with Ajax

- Web application: a dynamic web site that mimics the feel of a desktop application
  - Presents a continuous user experience rather than disjoint pages
  - Examples: Gmail, Google Maps, Google Docs, Flickr
- Ajax: *Asynchronous JavaScript and XML*
  - Not a programming language; a particular way of using javascript
  - Download data from a server in the background
  - Allows dynamically updating a page without making the user wait

    * Avoids the “click-wait-refresh” pattern

- Asynchronous: user can keep interacting with page while data loads
    - Communication pattern made possible by Ajax

Typical Ajax Request

1. User clicks, invoking an event handler
2. Handler’s code creates an XMLHttpRequest (XHR) object
3. XHR object requests page from server
4. Server retrieves appropriate data, sends it back
5. XHR object fires an event when data arrives
    - Often called a callback
    - You can attach a handler function to that event
6. Callback event handler processes the data and displays it

XMLHttpRequest (XHR) Object

- XMLHttpRequest object
    - Resides on the client
    - Layer between the client and the server that manages asynchronous requests in Ajax applications
- To initiate an asynchronous request
    - Create an instance of the XMLHttpRequest object
    - Use its *open* method to set up the request, and its *send* method to initiate the request
- For security purposes the XHR object does not allow a web application to request resources from domains other than the one that served the application
    - Known as the Same Origin Policy (SOP)
    - Can be overridden by setting CORS (Cross Origin Resource Sharing) headers (on the server you’re requesting from) to explicitly allow requests from another domain

The WebSocket Protocol

- AJAX always has to poll the server for data rather than receive it via push from the server

  - If you're moving a high volume of data - overhead of creating a HTTP connection every time is going to be a bottleneck

- WebSockets allow your client-side JavaScript to open and persist a connection to a server
  - With WebSockets, data is exchanged as messages, which can happen very quickly due to the persistent connection
- Another powerful aspect of WebSockets is capability called full duplex
  - Contrast with AJAX where the server has no method for pushing messages to the client

How WebSockets Work

- WebSocket specification defines an API establishing a two-way "socket" connections between a web browser and server
  - Persistent connection between the client and server
    * Both parties constant sending data at anytime
- The client establishes a WebSocket connection through a process known as the WebSocket Handshake
  - Process starts with the client sending a regular HTTP request to the server
  - *Update header* is included in the request
    * Informs the server that the client wishes to establish a websocket connection

WebSocket Efficiency

- With WebSockets you can transfer as much data as you like in both directions simultaneously
  - Without incurring the overhead associated with traditional HTTP requests
- Data is transferred through a WebSocket as *messages*
  - Each of which consists of one or more frames containing the data you are sending (the payload)
  - 2-byte cramming overhead
- Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred
  - Leading to significant reductions in latency

WebSocket API

1. Open a secure WebSocket connection (wss)
2. Optional callback, invoked if a connection error has occurred
3. Optional callback, invoked when the connection is terminated
4. Optional callback, invoked when a WebSocket connection is established
5. Client-initiated message to the server
6. A callback function invoked for each new message from the server
7. Invoke binary or test processing logic for the received message