Course Notes

EECS 485

Web Databases & Information Systems

Andrew DeOrio, H. V. Jagadish - Fall 2015

Contributors: Max Smith

Latest revision: September 28, 2015

Contents

Abstract

This course is a contemporary exploration of modern webbased information systems. It will integrate concepts from multiple computer science topics used in the design, development, and deployment of webbased applications, services, and knowledge systems. While broad in scope, it will also cover several key concepts in depth, including: web networking protocols, web databases and applications, web services, web search, webrelevant security issues, web infrastructure, and webrelevant data mining. Students will learn how to incorporate these concepts into an engineering process that includes design, analysis, development and testing, using technologies such as HTTP, XML, JavaScript, AJAX, and others.

1 Introduction and Overview

– See the syllabus.

2 Web Basics

2.1 Networking Basics

– **Circuit switched**: a dedicated channel is established for the duration of a transmission.

  – Good for real-time things like voice and teleconferences
  – Don't have to packet data
  – Better latency
  – No waittime

– **Packet switched**: netwrok in which relatively small units of data called **packets** are routed through a network for transmission.

  – Makes better use of network resources
  – Can survive destruction of a node in the network
  – Packets can get lost

2.2 Network Protocol Stack Model

Application	User interaction	HTTP, FTP, SMTP
Presentation	Data representation	XML, Cryptography
Session	DIalogue management	???
Transport	Reliable end-to-end link	TCP
Network	Routing via multiple nodes	IP
Data Link	Physical addressing	Ethernet
Physical	Metal or RF representation	802.11, Bluetooth

– IP is best-effor; therefore, packets may get dropped or delayed

– TCP is reliable because it guarantees data will get there in order

– Open System Interconnection (OSI) Reference Model

–

2.3 HTTP Structure

- Request/response protocol:
  1. Client opens TCP connection to server and writes a request
  2. Server response appropriately
  3. COnnection is closed
- Completely stateless
  - Each request is treated as brand new
- Client requests have several possible forms:
  - GET, POST, PUT, DELETE, HEAD, TRACE, CONNECT, OPTIONS
  - Each has an associated parameter
- Even a single page can consist of dozen of HTTP requests

2.4 HTTP Client Algorithm

1. Wati for user to type into browser
2. Break the URL into host and path
3. Contact host at port 80, send GET ¡path¿ HTTP/1.1
4. Download result code and bytes
5. Send content bytes to HTML renderer for drawing onscreen

2.5 HTTP Server Algorithm

1. HTTP server process (or thread) waits for connection from client
2. Receives a GET /index.html request
3. Looks in content directory, computs name /content/index.html
4. Loads file from disk
5. Write resposne to clinet: 200 OK, followed by bytes for /content/index.html

2.6 URL Encoding

http://server:port/path#fragment?search

- Server name translated by DNS look-up: www.umich.edu→135.22.87.1
- Path is a file name relative to server root
- Fragment is identified at the client, ignored by server
- Search string is a general-purpose (set of) parameter(s) that the server can use as it pleases.

3 Networking I: TCP/IP

3.1 Traceroute

- `traceroute` uses debug messages to expose packet's route to destination

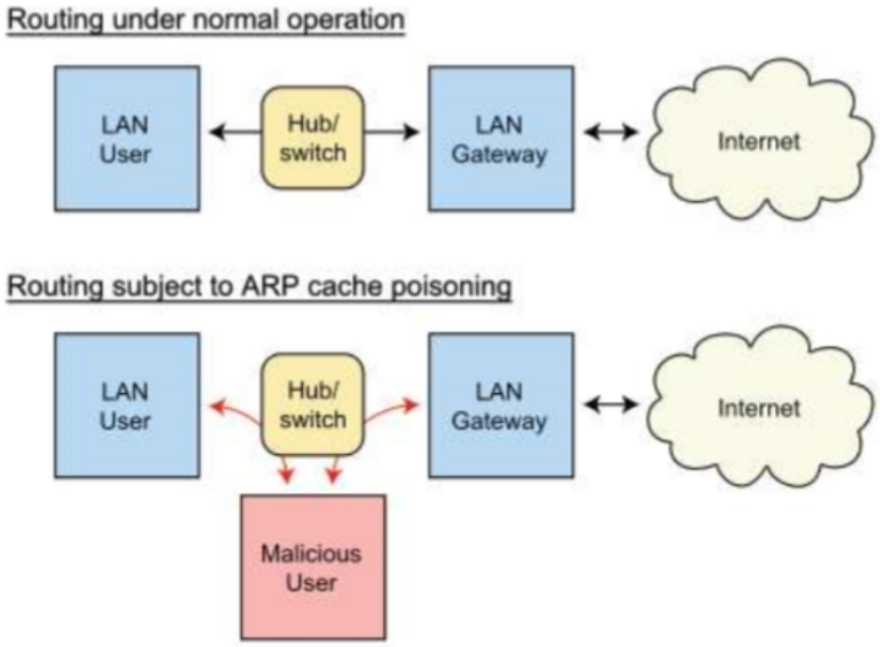
```
traceroute to google.com (74.125.95.99), 64 hops max, 52 byte packets
1 141.212.111.1 (141.212.111.1)  1.037 ms  0.681 ms  1.020 ms (Ann Arbor, MI)
2 13-caen-bin-arb.r-bin-arbl.umnet.umich.edu (192.12.80.177)  0.566 ms  0.378 ms  0.328
  ms(Ann Arbor, MI)
3 3  13-barb-bseb-2.r-bin-seb.umnet.umich.edu (192.12.80.11)  0.464 ms  0.495 ms  0.485
  ms(Ann Arbor, MI)
4 4  v-bin-seb-inet-aa2.merit-aa2.umnet.umich.edu (192.12.80.37)  0.749 ms  1.305 ms  0.773
  ms(Ann Arbor, MI)
5 6  207.72.112.46 (207.72.112.46)  6.552 ms  6.609 ms  6.495 ms(Ann Arbor, MI)
6 7  216.239.48.154 (216.239.48.154)  6.785 ms  52.824 ms  6.722 ms(Mountain View, CA)
7 8  209.85.241.22 (209.85.241.22)  43.101 ms  17.464 ms  17.575 ms(Mountain View, CA)
8 9  209.85.241.29 (209.85.241.29)  17.603 ms(Mountain View, CA)
9 10  72.14.239.193 (72.14.239.193)  18.306 ms  25.136 ms(Mountain View, CA)
10 11  iw-in-f99.1e100.net (74.125.95.99)  18.390 ms  18.194 ms  18.149 ms(Mountain View,
  CA)
```

- No reason to have the packet links to make sense (they're not based on geography, only connectivity)
- e.g.,
  - Ann Arbor, MI
  - San Mateo, CA
  - New York, NY
  - ¡ends¿

3.2 IP's Small Corners

- How does a host get an IP address
  - Used to be static
  - Nowadays, often dynamic - DHCP (Dynamic Host Configuration Protocol)
- How does a host get its gateway address (to chat from local to internet)?
  - Dynamically from Router
  - Originally static each get their own wire

3.3 ARP



- How does an IP address get mapped to a target ethernet address?
  - Address Resolution Protocol (ARP)
- This leaves the possibility of ARP spoofing (or poisoning) where you pretend to be another address
- **ARP spoofing**
  - Associate attacker's MAC address with IP of another host(s)
  - Man-in-the-middle attacks
  - Reconfigures table switch
  - Switch only sends to destination
  - Hub sends to everyone
  - CPU ignores not intended for your CPU (spoofing overrides this)

3.4 Domain Name Service

- How does a browser known which IP address to contact?
- BY a directory look-up, using **Domain Name Service (DNS)**
- Cache recently used domain names to reduce need for DNS look up
- e.g.,

```
1 $ host umich.edu
2 umich.edu has address 141.211.243.44
```

- Host checks for name mapping in DNS (umich→IP)

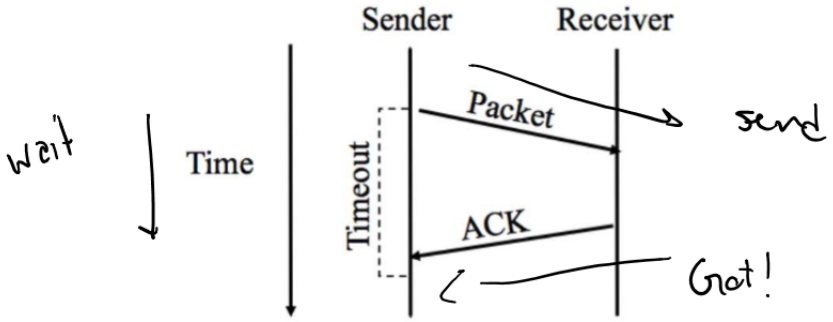
3.5 TCP

- Transport Control Protocol
- Reliable, ordered bytestreams
- Connection-oriented, unlike IP
- "Virtual circuit" networking built on packet infrastructure
- Looks like a circuit, but no reservations (on IP)
- Processes tied to "host:port" pairs
  - Packets/processes talk to ports
  - Ports are OS-level concept
  - Server ports are "well0-known" and associated with services; other ports are "ephemeral"
- Common TCP ports:
  - HTTP - 80
  - SSH - 22
  - HTTPS - 442
  - SMTP - 25
  - IMAP - 143
- Message stream is bidirectional

3.6 Implementing Connections

- Basic principle of reliable TCP connection is retransmission
  - Each packet has 32-bit sequence number
  - Every SeqNo is ACKed (acknowledged) by receiver
  - When timeout expires, sender emits again

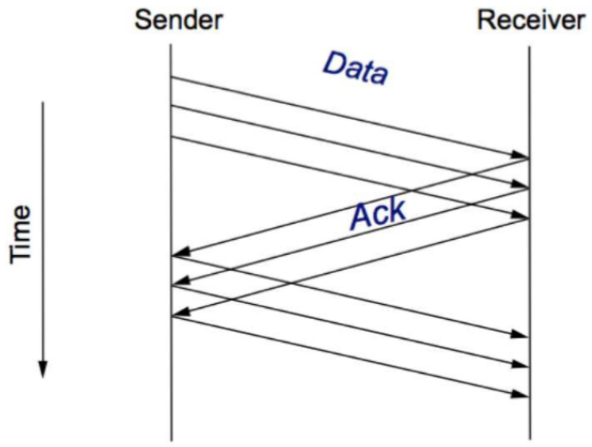
3.7 Stop and Wait



- Send a packet, wait for ACK
- Receiver ACKS everything
- Are there downsides to Stop-and-wait?
  - Never get a resposne back

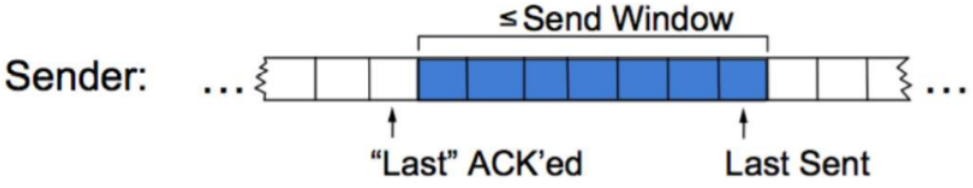
3.8 Sliding Window

- **Sliding window** technique for managing send/receive capacity



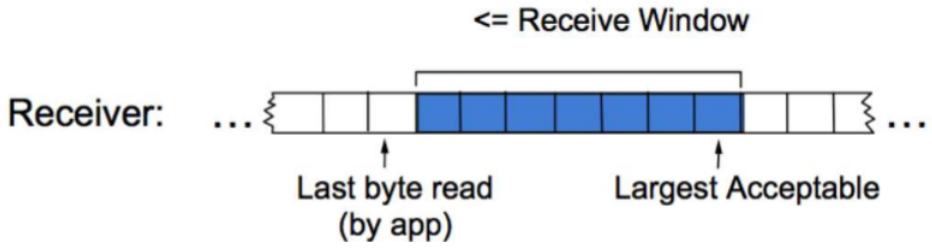
  - Receiver indicates "receive window" it is willing to buffer
  - Sender cannot have more packets in transit (unACKed) than what can fit in the window
  - Ideally, window is bandwidth * RT delay
    - Keeps as much data in transit as possible
- This algorithm has several roles:
  - Reliable delivery, via ACKs, timeouts, and retransmissions
  - In-order delivery, via buffering and ACKing
  - Flow control, via advertised window for receiver
- Problems with sliding window?
  - What if sender trnasmits data too fast?
  - Or receiver reads data too slow?

3.9 Sliding Window - Sender



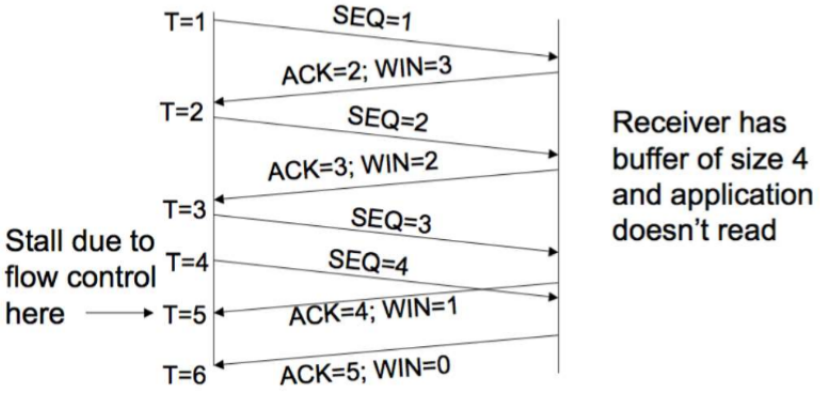
- Sender buffers unACKed data
- Only removes data from send buffer after it's been ACKed
- Send window determined by receiver's advertisements

3.10 Sliding Window - Receiver

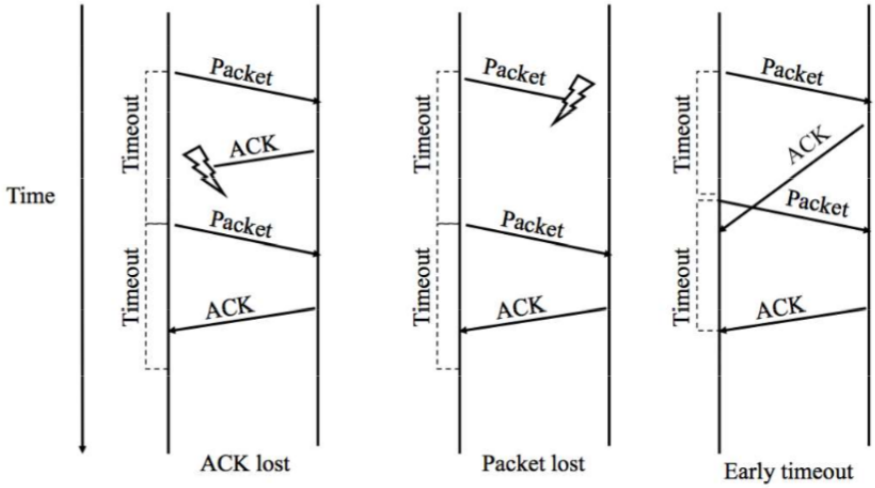


- Receiver buffers data that is
  - Out-of-order
  - Not yet read off by application
- ACKs data as it arrives
- Removes data from buffer as app reads
- Also shrinks/expands advertised window in response to application behavior

3.11 Example

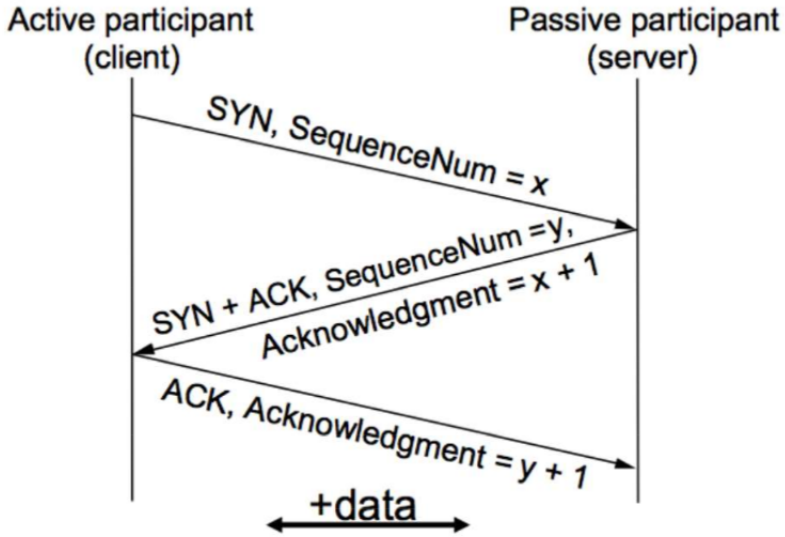


3.12 Errors



- **ACK lost**: ACK was lost from receiver to sender, so the sender doesn't know that the receiver got the packet. Sends it again anyways!
- **Packet lost**: Receiver never got packet, so sender never gets ACK; therefore, it tries again!
- **Early timeout**: If the timeout is too short, the sender will send the same packet again before the ACK had time to come back. This just results in a waste of resourdces

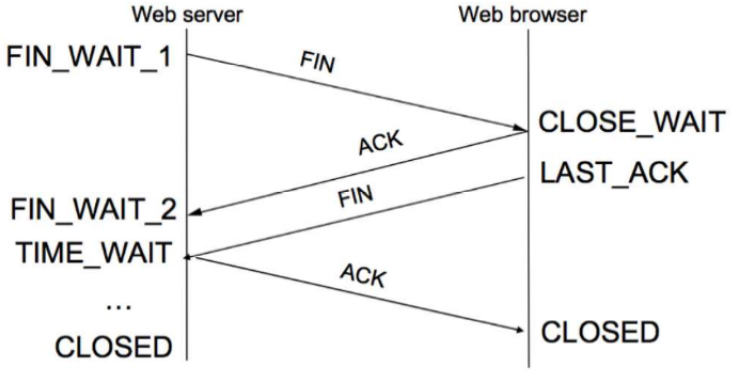
3.13 3-way Handshake



3.14 Connection Teardown

- Orderly release by sender + receiver
- "Hanging up the phone"

- Releases states and buffers on both sides

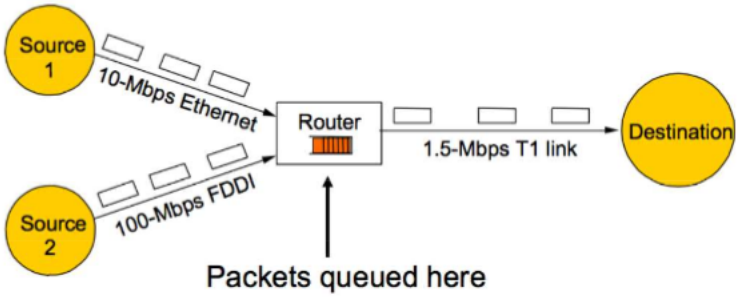


3.15 Slowing Down

- Two reasons for sender to slow down:
  - Reeiver can't handle input (and will have to drop packets)
  - Network can't transmit sas fast as incoming packets arrive
- Sliding window algorithm takes care of the receiver
- Sender never transmits more than advertised window size

3.16 Congestion

- Buffers in middle of network can become overloaded
- Not part of window negotiation (not accounted for)

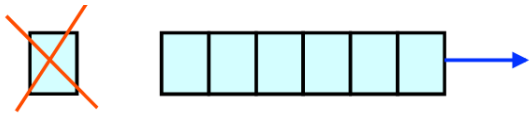


- When has a packet been lost?
  - Too long a timer, you're waiting pointlessly
  - Too short, adds needless load
- Many retransmits can induce **congestion collapse**
  - Retransmits just add to congestion
  - Capacity of network falls dramatically
  - Increase in load that results in a decrease in useful work done

Where It Happens: Links

- Simple resource allocation: FIFO queue and drop-tail
  - Access to the bandwidth: FIFO queue
    - Packets transmitted in the order they arrive

  - Access to the buffer space: drop-tail queueing
    - If the queue is full, drop the incoming packet



End Host

- Packet delay: packet experiences high delay
- Packet loss: packet gets dropped along the way
- How does TCP sender learn this?
  - Delay: round-trip time estimate
  - Loss: timeou, acknowledgements

3.17 TCP Congestion Window

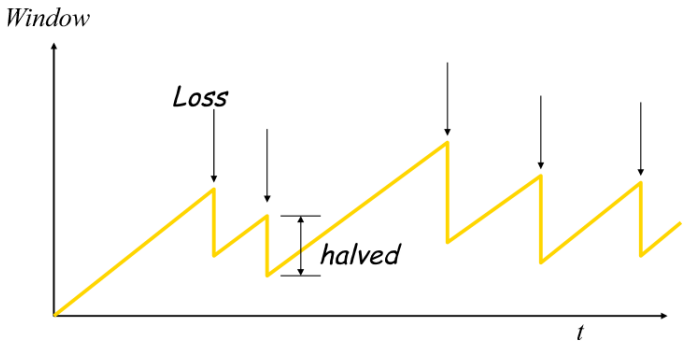
- Each TCP sender maintains a congestion window
  - Maximum number of bytes to have in transit
  - Number of bytes still awaiting acknowledgments
- Adapting the congestion window
  - Decrease upon losing a packet (backing off)
  - Increase upon success (optimistically explorining)
  - Always struggling to find the right transfer rate

3.18 Additive Increase Multiplicative Decrease (AIMD)

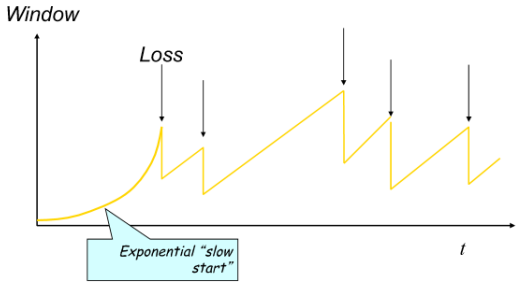
- AIMD algorithm:
  1. After packet timeout, cwnd = cwnd/2
  2. If none, cwnd += 1 every RTT
  3. Sender always xmits min(win, cwnd)
- Why use AIMD? Network load is really hard to get rid of: oversubscribed link must be undersubscribed to dissipate queue

- Over: packets dropped and retransmitted
- Under: somewhat lower throughput

AIMD Sawtooth



- The previous diagram has a slow-start, so we can implement exponential window size to begin with to increase starting speed



3.19 HTTP 1.0

- Every HTML-embedded item requires a GET (index.html, ad.gif, logo.gif, etc.)
- Naive HTTP on TCP yields awful performance:
  - Many TCP connections created roundtrip
  - Lots of slow-start delays

3.20 HTTP 1.1

- Persistent Connections
  - Server does not close the connection after sending the resposne
  - Client can re-use it
    - Especially improves small objects
    - Makes parallel downloads difficult
- Pipelining

  - Many HTTP requests can be “live” at once, on the same persistent connection
  - Send lots of HTTP request at once, then get lots of answers
  - Server replies in the order received
- Caching
  - GET + IF-MODIFIED-SINCE ¡timestamp¿
  - When combined with browser cache, eliminates a lot of unneeded data transfer
  - Same number of roundtrips
  - Lots of different caches possible

4 Content and Dynamism

4.1 Mark up Language

- Add tags as “mark up” to text
- Document still “primarily” text
- Mark up improves both structure and presentation

Hypertext

- Text with embedded links to other documents
- Anchor tag ¡a href = “link”¿atext¡/a¿

Escape Strings

- Some characters have special meaning in an application context
  - / or ? in URL
- Need an **escape string**
  - e.g., < € &

4.2 XML

- No standard set of stags
- Define tags and use them
- Need to balanced like parentheses
- Tags form a heirarchy of objects
  - Document Object Model (DOM) tree

4.3 XHTML

- HTML permits unbalanced tags
- HTML cannot be derived from XML
- XHTML is a dialect of HTML that meets XML rules (proper containment)

4.4 HTML5

- Merges all that has happened over years related to HTML
  - XHTML
  - Browser-specific estensions of HTML
  - OTher use cases of broad interest

4.5 Layout - CSS

- Modern HTML seperates content form the way content looks (through CSS)
- Included via:

Scaling and Positioning

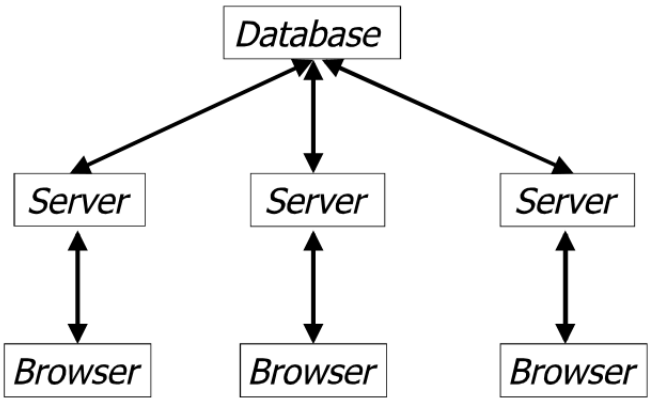
- Biggest problem in layout
- If display screen width changes, do you:
  - Keep your page fixed width any way
  - Adjust page width, and use more lines of text - increasing page length
  - Adjust path width and font size so everything scales evenly

4.6 Dynamic Content

- So far, HTTP servers are fileservers
  - And browsers are HTML renderers
- Think of the things that are impossible with simple static pages
- In the old days, almost all requests were just disk loads

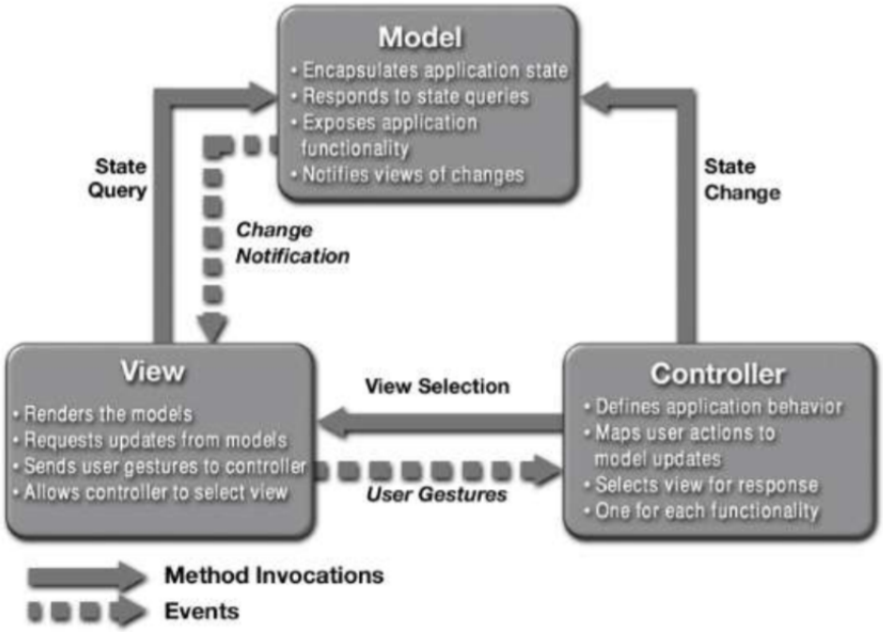
4.7 N-tier model

- We've seperated persistent storage from user interactions
- Web apps often break down pieces of code functionality into machines

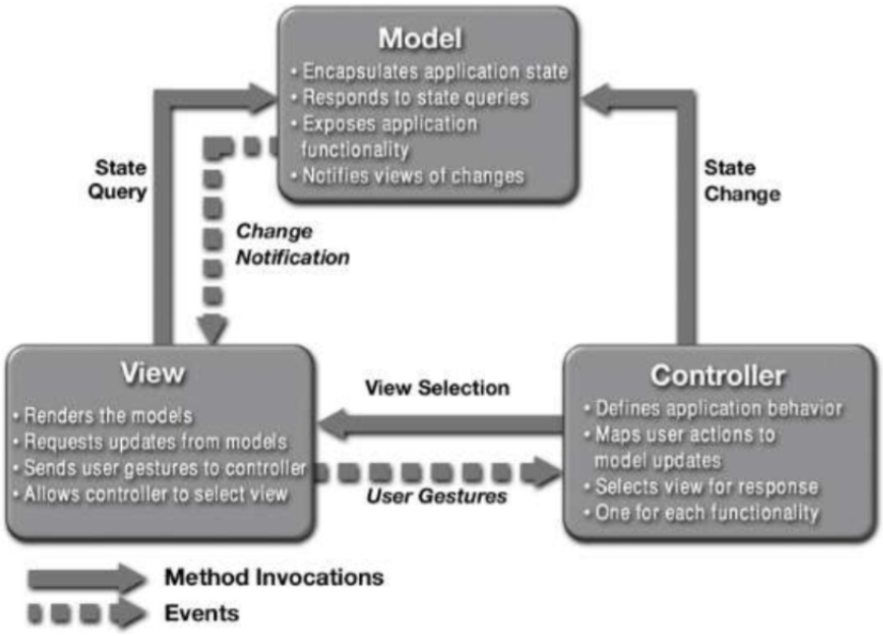


Model-View-Controller

- System to sort out who does what



- HTML is the model
- CSS is the view
- Browser is the controller
- Model exists in the database/filesystem
- View is the computed HTML++
- Controller is processing of user input

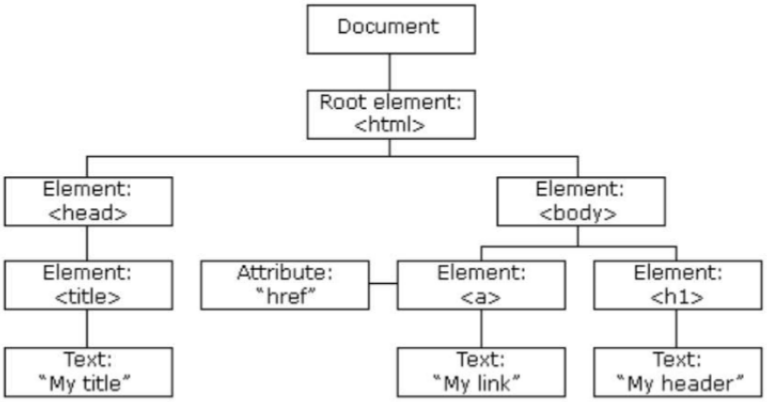


4.8 Object-Relational Mapping

- Addresses the "impedance mismatch" between relational and object worlds
- Map object classes to database tables

4.9 Dynamic Client

- Lots of different options for client-side content, but most people nowadays conerging on AJAX (Asynchronous JavaScript and XML)
  - JavaScript for running in the browser
  - XML for data exchange with server, via HTTP
  - After intiial page load, similar to client/server program, not traditional web page loads
- JS code runs when triggered by
- Has read/write access to the page's DOM
- e.g.,



– Client-side code will display code instead of end result. Server-side code would only show result.

4.10 Browser Security

– JS is potential security nightmare, so in-browser programs are sandboxed

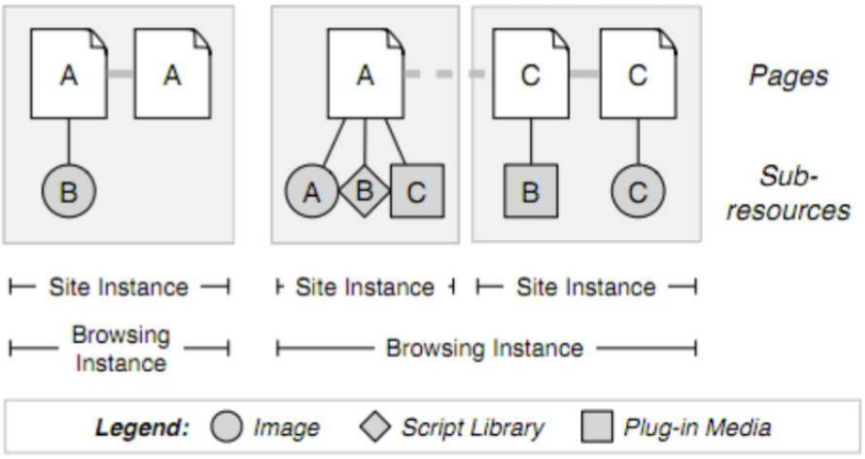
  – **Same-origin policy**: scripts from the same origin can access each others' data + methods
  – scripts form different origins cannot see each other

Attacks

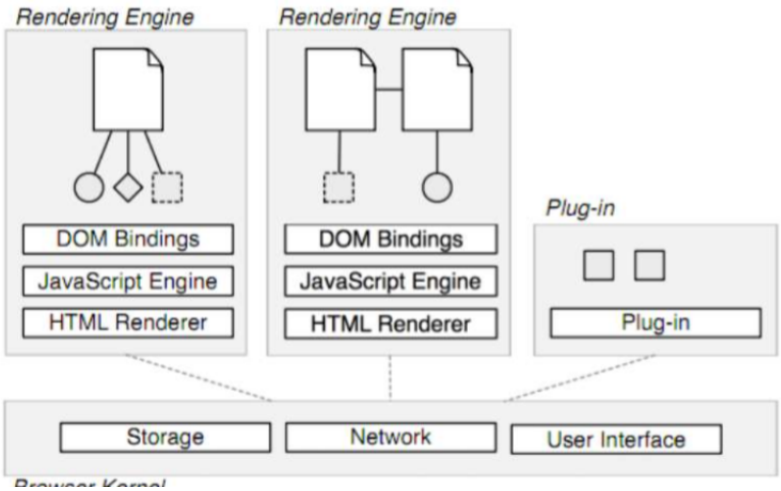
– Cross-site scripting

– SQL Injection

4.11 Browser Internals



– Chome added more break-down of functionality allowign you to kill tabs-by-process

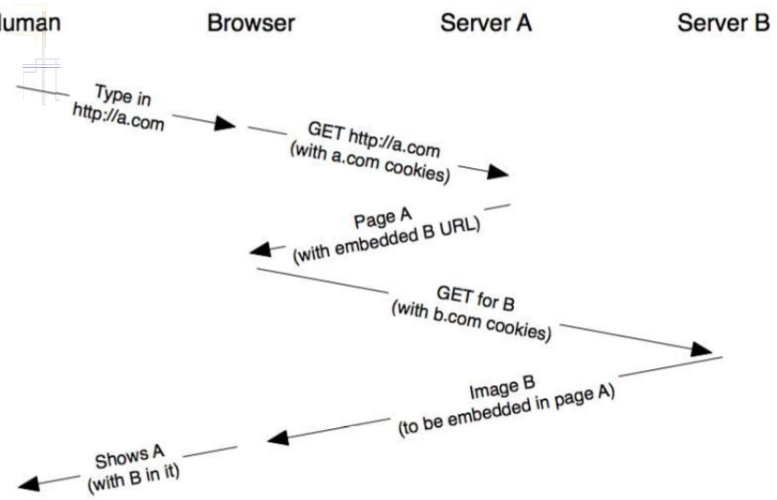


5 Sessions

5.1 Cookie Contents

- **Path**: spedifies scope of cookie
- **Expiration**: tells client when to delete
- **Secure**: is how ocokie may be transmitted
- Contents up to server are encrypted
- Supported by HTTP
- Overwriatble by source domain and path-prefix
  - This stops a site from overwritting a different site's cookie
- Browser per-domain, total limits
  - Each website only gets so many cookies and so much data
  - Badly designed websites with large cookies may not get totally stored

5.2 Uses and Abuses



- Browser makes more HTTP requests then you think (1 probably has additional 10 hidden)
- The cookies for page B will remember all browsing history through various pages (e.g., A) so they can accumulate your history and your behavior
- Cookies make sessions quiet, ubiquitous

6 Encryption

6.1 Encryption as a Function

- Plaintest string s
- Encryption key K_{enc}
- Decryption key K_{dec}
- Encrypt s with K_{enc} to obtain ciphertext $K_{enc}(s)$
- Decrypt $K_{enc}(s)$ with decryption key K_{dec} to reobtain s
- $K_{dec}(K_{enc}(s)) = s$
- Encryption applies a reversible fn to some piece of data, yielding something unreadable
- Decryption recovers the original data from the unreadable encryption-output
- The encryption/decryption algorithm assumed known; the key is secret

6.2 Substitution Ciphers

- Arbitrary 1:1 mapping of alphabet chars, using a **substitution table**
- All of these are vulnerable to frequency analysis:
  - letter
  - word
  - common phrases
- We could make the substitution table larger (translating n-gram, not chars):

Plaintext	Ciphertext
AAA	QWE
AAB	RTY
AAC	ASD

- How big is substitution table?
  - A^n entries, where A is size of alphabet and n is size of grams
- Still vulnerable, but requies more text

6.3 Substitution Rules

- Don't store table explicitly; derive table rows using substitution rule
  - e..g, sXORk, where k is the key
  - Remember: security level depends on size of key
  - Key of len $b \geq 2^b$ possible keys

- XOR "flips a bit" for input bits that correspond to key's "1"
- Encrypted string should ideally show no pattern for frequency analysis attack
- What's the right key size?
  - Who is trying to break the scheme?
  - Is that good enough?

6.4 Data Encryption Standard (DES)

- DES is a block cipher with 56-bit key
- Data transmitted in 64-bit blocks, each may be coded independently
- Triple-DES, breaks up text in 3-56 bit chunks and apply DES to each with different keys

6.5 Review of Crypto

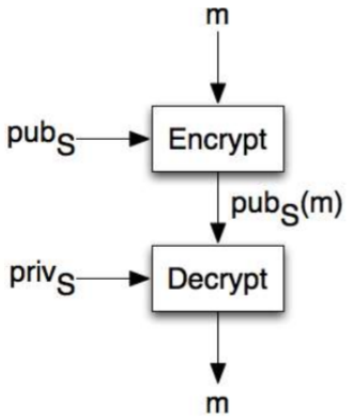
- The key is traditional crypto is used to encode the substitution rule
  - Needed to encrypt and decrypt
- Key distribution is the weak link
  - Hard to revoke
  - Disastrous if "codebook" is compromised
  - Hard to distribute
  - Impossible for the Web

6.6 Public-Key Cryptography

- Secure communication without key exchange
- Each party has a pair of related keys: **public** and **private**
  - A public key is published freely
  - A private key is shared with no one
- A message enrypted witho ne can be decrypted with the other
- YOu can't derive one from the other (this is critical!)

Two Modes

- I encrypt using the public key, and decrypt using the private key



- Anyone can encrypt; only S can decrypt
- Used for data confidentiality
- If encrypted using private, anyone with public can decrypt
- Used for authenticity
- So if someone was able to encrypt, then they must have had private, ensuring no man-in-the-middle

6.7 Trapdoor Functions

- Public key cryptography relies on so-called trapdoor functions
  - A fn that is easy to compute, but hard to invert without special info
  - "easy" and "hard" meant comutationally
- Some poor choices: add, multiply
- In practice quite difficult to find good trapdoor rfunctions
- Most popular one is related to prime factorizaiton; others possible

Prime Factorization

- $n = pq$, where p and q are primes
  - Given p and q, easy to compute n
  - Given n, very hard to find p and q

Fermat's Little Theorem

- For any prime p, and any integer a: $a^p = a(\mod p)$

Chinese Remainder Theorem

- Consider $x = a_i(\mod p_i),$ for $i = 1, \ldots, k$
- CRT: There's a solution for x if p_i are pairwise relatively prime (i.e., have no common factors greater than 1)
- If all a_i are 1, then $x = 1(\mod p_i)$

6.8 Cryptography

- We choose two large primes: p, q
- $n = pq$
- Next:
  - Set $\lambda = (p-1)(q-1)$
  - Choose e randomly, such that $e < \lambda$
  - Choose d, such that $de = 1(\mod \lambda)$
- n and e serve as public key
  - n is product of primes p, q
- n and d serve as private key
  - Choosing d requires e and λ
- $\text{encrypt}_{n,e}(m) = m^e(\mod n) = c$
- $\text{decrypt}_{n,d}(c) = c^D(\mod n) = m$
- Slower than DES because exponential instead of XOR or SHIFTs
- Will decryption always work?
  - $c^d = m^{de} = m^{k\lambda+1}$
  - $m^{k\lambda+1} = m(m^{(p-1)(q-1)})k$
  - Recall from Fermat that:

    - $m^{p-1} = 1(\mod p)$
    - $m^{q-1} = 1(\mod q)$
  - Which implies:
    - $m^{(p-1)(q-1)} = 1(\mod p)$
    - $m^{(p-1)(q-1)} = 1(\mod q)$
  - Use the CRT to combine above 2 equations:

$$m^{(p-1)(q-1)} = 1(\mod n), \text{ where } n = pq$$

  - $c^d = m(1)^k(\mod n)$
  - $c^d = m(\mod n)$
  - Thus, decrypted ciphertext $c = \text{msg}(m)$
- Send private shared keys via expensive method, then use cheap method once key is shared. HTTPS does this.