COMP 307 FALL 2018 - FINAL EXAM NOTES [Margaret Gu]

HISTORY OF THE INTERNET

Stand-alone custom program:

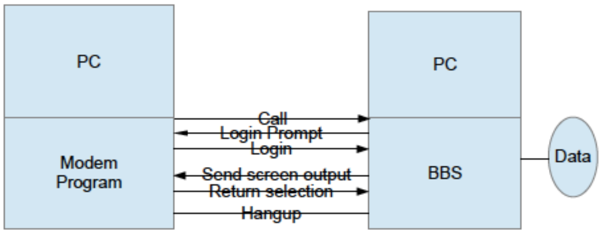
- Home computer > modem > telephone > PBX(as router) > Server

Before the internet:

- **Modem**: converted software data into sound and transmitted over tele wire
- **Packets**: a packaged set of data
- **Phone numbers**: used for routing; a dedicated connection

BBS(Bulletin Board Service)

- Remote connection with modem to server
  - Shows a window to their server, no execution on client end
- Modem and BBS interfaced with PC hardware directly: little need for OS and server
- Use OS for read/write



Front-end: transmit user selection to backend

Back-end: process user requests

API to control communication between front and back. (Ex. SSH give u a lot of control over backend)

EVOLUTION OF THE INTERNET

Beginning of Internet

- July 1961 - Leonard Kleinrock: theory of packet based networks
- August 1962 - JCR Licklider: theorized a Galactic Network
- 1965: low speed dial-up connection; first wide-area network
- ARPANET (Advanced Research Project Agency Network) first network
- ARPANET first > internet > NSFNET > WWW > internet2
- Archie: first true internet search engine; developed @McGill
- Mosaic: early browser

Network Protocols LECTURE

- TCP/IP: Transmission Control Protocol/Internet Protocol

4 Internet Ground Rules (IMPORTANT MEMORIZE)

1. Each distinct network should stand on its own
2. Communication tries its best - fail to receive packet = retransmit again later
3. Black boxes (gateway and routers) used to connect networks
4. No global control at operational level

Stack

- Set of applications that work together as a software system to **connect front-end and back-end architectures**
- XAMPP, MEAN, Django
- Common stack example
  - Client: HTML CSS JS
  - Communication: CGI JSON
  - Server: REST server, PHP, SQL

BASIC NETWORK ARCHITECTURE

Network

- A collection of machines connected with a medium, which passes information of a particular **format and protocol**
- How do networks control the flow of info? Use formats and protocols

Simple Peer-to-Peer Network (no server)

- Mediums can be: Wire, Radio, or Light

Simple Network with Server

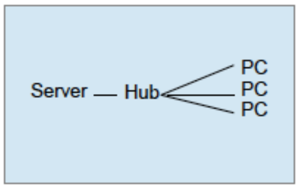
- A wants to send message to B, msg must pass through server.
- Server has 3 options:
  - Pass info onwards
  - Execute on server
  - Make a backup
- How does a machine uniquely identify itself? With an **address** which is an int

Ring network
- Ex. Peer-to-peer
- Data travels in one direction so data has to travel through intermediate devices
- Best to be the connection right before server, so ur packets are only seen by server
- Security depends on how close u are to server

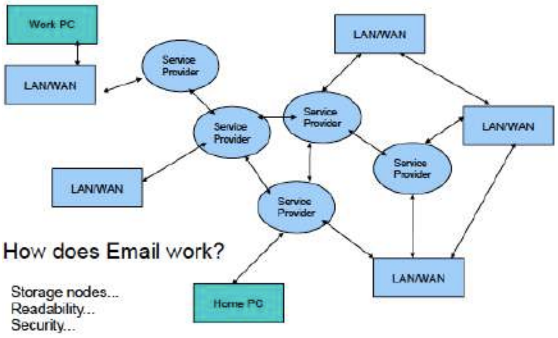
Star network (ex. Wifi)
- Most secure bc everybody is *directly* connected to server; expensive

Backbone Network
- Everybody is connected to one wire so everybody sees packet
- Cheap so common network

Tree Network
- Central connection isn't a server; they're hubs and routers
- Star connection for each hub with its users, so still secure



Network Components:
- **Wire**: a medium that info travels through
- **Network card/modem**: convert strings to signals compatible with medium
- **Hub or Router:** splits input wire into multiple output wires
    - **Hub**: for broadcasting
        - Has no intelligence and floods/broadcasts packets to connected devices
        - Use if server wants to send packet to its connected devices
        - data can be passed through hub and not server, relieves some work for server
    - **Routers** pick a path
        - Has memory and does not need to flood
- **Server**: responsible for managing communication and sharing between connected devices
    - **ISP**(internet service provider): a server responsible for providing Internet, routing, addressing and traffic control
        - Has members
        - Provides URL to IP address
        - Routing tables to calculate shortest path or next best hop
        - Can broadcast by choice or when its dumb
        - Can do simple load balancing and traffic avoidance
    - **Bridge**: server that translates packets to diff formats; allows for communication between networks
        - Smart and checks for permission to pass packets
        - Between hubs and can stop flooding of packets to next hub
    - **Gateway**: server that provides connection with local area network (ex. ISP)



How does Email work?

Storage nodes...
Readability...
Security...

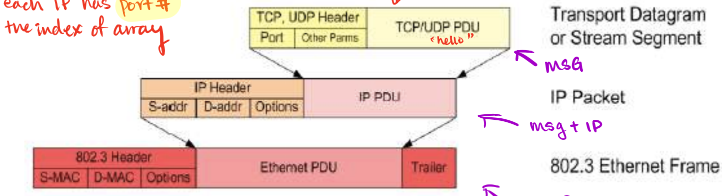
PACKETS

Packets
- Data converted into shit (music, ex. Wifi)
- Packet contains: src and destination address, msg and error correction bits
- Packet syntax as string: addr:addr:msg:correction
    - Initial state is unencrypted
- Data structure used to store data for transmission
    - Payload: actual msg
    - Fragment offset: tells u the packet # when ur msg is sent through multiple packets

port #: tells OS which IP is for
↳ each IP has port #
↳ the index of array

Packet elements:
- Addresses
    - MAC: media access control; physical and unique
    - IP: internet protocol; logical and can change/set
- Data: ASCII to
    - binary for wave
        - 2 frequencies: 1 and 0
        - 4 freq: highest: 11, 10, 01, 00 : lowest
        - 8, 16 etc free: sensitive equip needed, expensive

ex. when printf("hello")
what users/programmers can control



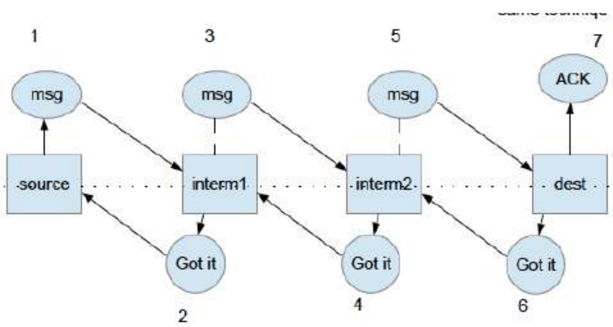
MSG

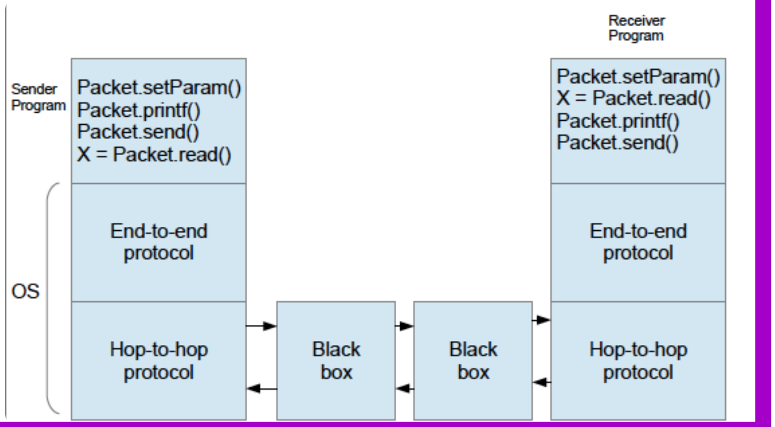
msg + IP

msg + IP + MAC
Convert to actual signals for transmission

.Important picture! NESTED STRUCTURE
- Bottom: The frame with control information
- Middle: The packet with control information
- Top: The message with control information

        - There is music before the actual msg signal to sync for proper reading of signal
    - Modem and network cards' tasks
        - Convert str to binary(wave)
        - Modulate for speed
        - Broadcast transmission down wire/medium
- Protocols: an all for how to transmit data
    - End-to-end protocol (an application protocol)
        - Informs source and dest of status of a msg; does not care about route
        - A msg can be stored in several packets and identified by a **segment number** b/w the packets
        - **Algorithm**
            - **Source**
                - Try 3 times: send # of segments, wait for ACK or fail
                - Try 3 times: send a segment, wait for ACK, timeout? Resend
                - Terminate when all segments sent and ACK received
            - **Destination**
                - Wait infinite
                - On initial receive: check # of segments, send ACK or ERR, start wait timer again
                - Wait for segments: sort, store and ACK, timeout? Prompt (3 tries = ERR), Corrupt = ERR
    - Hop-to-hop protocol (a network protocol)
        - used bc there are many computers between src and dest devices
        - So packets must go through all the intermediate computers
        - This protocol figures out the traffic, lost and damaged packets; cares about route
        - Black boxes manage routing by sending **status packets** and are hidden from the application
        - confirmation packets are timed (max 3 tries)




Receiver Program

Sender Program

OS

Packet.setParam()
Packet.printf()
Packet.send()
X = Packet.read()

Packet.setParam()
X = Packet.read()
Packet.printf()
Packet.send()

End-to-end protocol

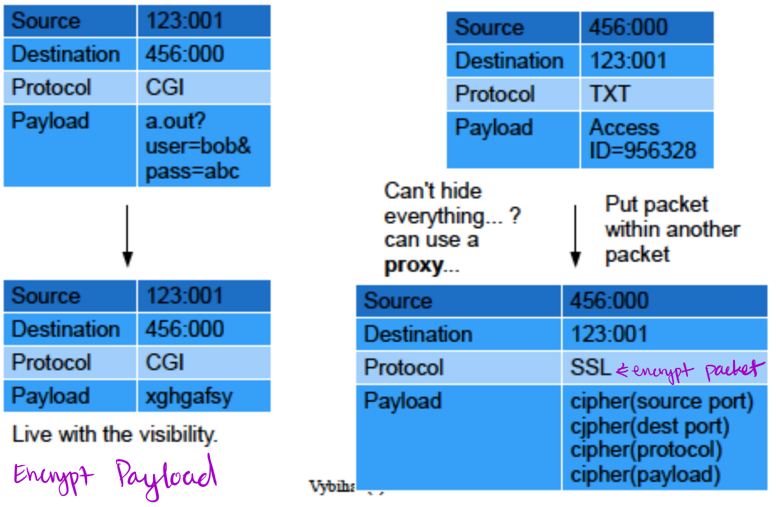
End-to-end protocol

Hop-to-hop protocol

Black box

Black box

Hop-to-hop protocol



Source	123:001
Destination	456:000
Protocol	CGI

Payload	a.out?
	user=bob&
	pass=abc

Source	123:001
Destination	456:000
Protocol	CGI
Payload	xghgafsy

Live with the visibility.

Encrypt Payload

Source	456:000
Destination	123:001
Protocol	TXT
Payload	Access ID=956328

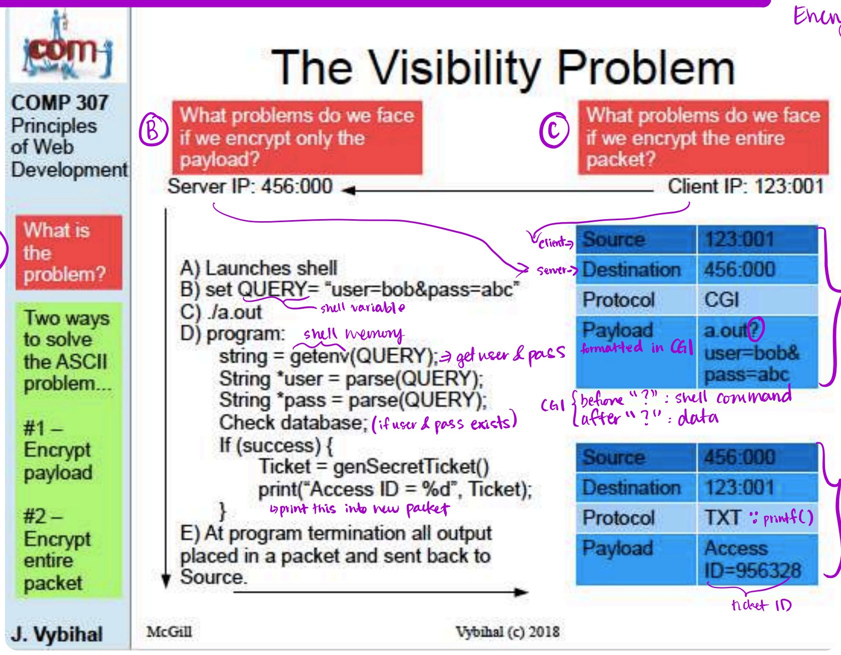
Can't hide everything... ? can use a **proxy**...

Put packet within another packet

Source	456:000
Destination	123:001
Protocol	SSL ⇐ encrypt packet

Payload	cipher(source port)
	cjpher(dest port)
	cipher(protocol)
	cipher(payload)

Vybihal



COMP 307 Principles of Web Development

The Visibility Problem

Ⓐ What is the problem?

Two ways to solve the ASCII problem...

#1 – Encrypt payload

#2 – Encrypt entire packet

Ⓑ What problems do we face if we encrypt only the payload?

Ⓒ What problems do we face if we encrypt the entire packet?

Server IP: 456:000 ← Client IP: 123:001

```
A) Launches shell
B) set QUERY= "user=bob&pass=abc"    shell variable
C) ./a.out
D) program:    shell memory
     string = getenv(QUERY); ⇒ get user & pass
     String *user = parse(QUERY);
     String *pass = parse(QUERY);
     Check database; (if user & pass exists)
     If (success) {
          Ticket = genSecretTicket()
          print("Access ID = %d", Ticket);
     }    ↳ print this into new packet
E) At program termination all output
placed in a packet and sent back to
Source.
```

client→ Source	123:001
server→ Destination	456:000
Protocol	CGI

Payload
formatted in CGI
	a.out?
	user=bob&
	pass=abc

packet sent to server

CGI { before "?" : shell command; after "?" : data

Source	456:000
Destination	123:001
Protocol	TXT ∵ printf()
Payload	Access ID=956328

ticket ID

response of server sent to client

J. Vybihal

McGill

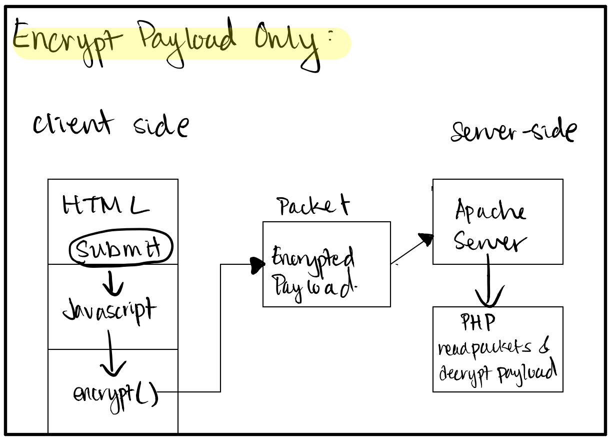
Vybihal (c) 2018

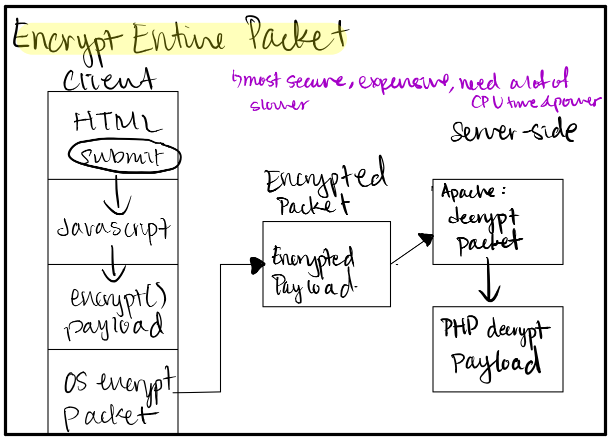
Ⓐ ⇒ everything is visible (know data, ur IP, who ur communicating w/)
↳ know ticket ID, so can log in as you.

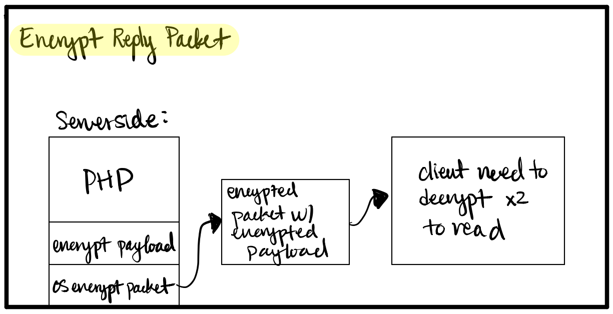
Protocol SSL encrypts packet

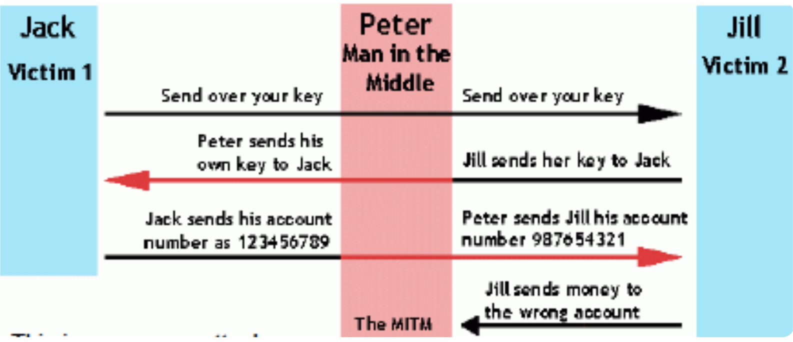
Ⓒ ⇒ how do we know where to send it? (expensive.
use hop & have shared key & saved as encrypted

Software POV






Encrypt Reply Packet
Serverside:
PHP
encrypt payload
OS encrypt packet
encrypted packet w/ encrypted payload
client need to decrypt x2 to read


Jack
Victim 1
Peter
Man in the
Middle
Jill
Victim 2
Send over your key
Send over your key
Peter sends his own key to Jack
Jill sends her key to Jack
Jack sends his account number as 123456789
Peter sends Jill his account number 987654321
Jill sends money to the wrong account
The MITM

Man-in-the-middle attack

- Trick the source and dest that bad guy is the src/dest
- Creates a new connection
- One-way attack in picture
- Two-way attack is to store Jacks key and send his (Peter) key to Jill in first step

Crytopgraphy delays bad guys

- Delay measured in computation resources and time
- 'Its secure bc it takes 100 years to break. Until then, I dont need to care."

CIPHERS

Substitution Ciphers (Reversible)

- Char replaced by another char, harder to read msg
- **Symmetric cipher**: sub ciphers that use a single key to encrypt and decrypt
    - Ex. For each letter, shift down two letters in the alpha (ex. A becomes E)

Caesar Cipher

- convert msg into ascii and add an int to ascii (since ascii is int)
- Send encrypted msg and number separately
- PROBLEM: we can match with the original letter frequency diagram of that language

Block Ciphers:

- Have multiple keys, used grouped together
- Ex. Key [2,3,1] and encrypt 3 letters at the time with this key. So A and D will be encrypted with 2
- Transposition: put msg in a table, and transpose it (flip the array)

DES: combo of symmetric and transposition; encrypt msg multiple times with encrypted versions of key (transpose)

Hash Cipher: (cant reverse) ↳16 times, not as effective/secure, but have 16 diff keys ... can be broken within powerful computer

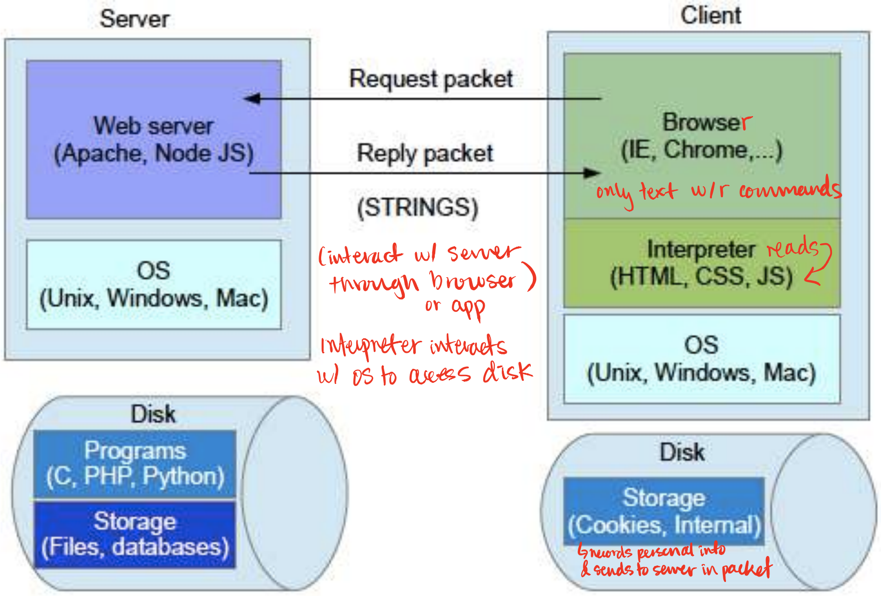
- Need: Secret key, Custom hash algorithm, Msg
- Msg reduced to a single hash number; Ex. Assign each letter a value, have an alg (ex. Sum of letter values)
- Problem: not unique

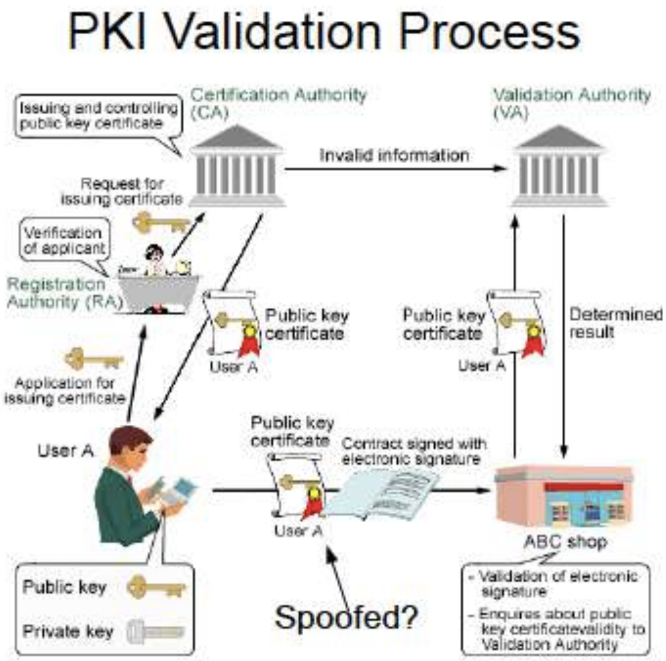
Asymmetric cipher: Cipher $= key^{(msg)}$ modu.

- 2 special keys (1 encrypt, 1 decrypt); modulo arithmetic and large prime #
- 1 public, 1 private key; gcd(modulo range, key) = 1 ⇒prime → larger prime = harder to crack.

Public key infrastructure (Certificate & Certificate Authority)

- Based on asymmetric cipher (1 private, 1 public key); certificate (file contain unique private public key pair)
- Used to encrypt/decrypt msgs, sign packets(signatures)
- X.509 Certificate, to ensure the key is valid and original




PKI Validation Process
Issuing and controlling public key certificate
Certification Authority (CA)
Validation Authority (VA)
Invalid information
Request for issuing certificate
Verification of applicant
Registration Authority (RA)
Application for issuing certificate
Public key certificate
User A
Public key certificate
User A
Determined result
User A
Public key certificate
Contract signed with electronic signature
User A
ABC shop
Public key
Private key
Spoofed?
- Validation of electronic signature
- Enquires about public key certificatevalidity to Validation Authority

Web Server

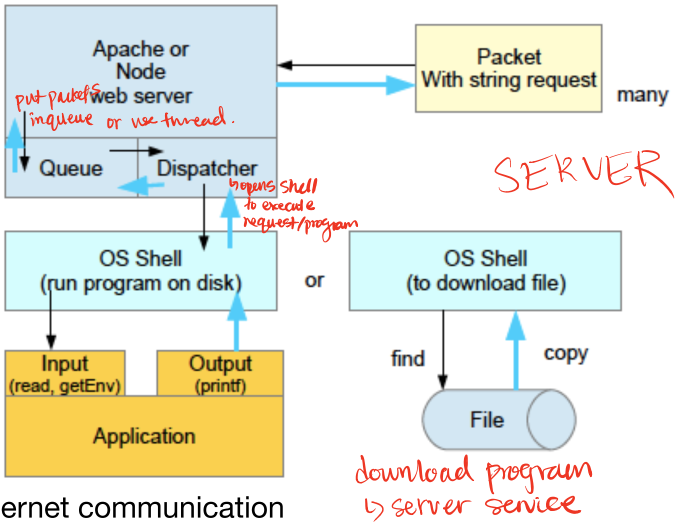
- Uses IP address for public addressing
- Uses port# for internal programs
- Server monitors port #
- When packet arrives at port#, it connects program with packet
  - STDIN = incoming packet data
  - STDOUT = future outgoing packet
  - Program's printf() writes to outgoing packet

UDP Protocol (speed)

- Source sends data to destination
- Source does not want ACK (∵ slow)
- Only uses hop-to-hop
- Ex. Email and streaming

TCP Protocol

- Uses both hop and end protocol
- The standard communication method for most network and internet communication



Handshake Protocol

- When one device wants to establish initial communicating with another device
- Protocol:
  - A. User sends hello packet
  - B. Host replies with protocol and encryption rules
  - C. User and host negotiate common rules
  - D. Host request for user and password
  - E. User provides user and ps
  - F. Host confirms or deny access

Cryptographic Authentication

- The challenge-response technique
- A secure handshake protocol
- If A and B are who they say they are, then the challenge should be encrypted as the challenger expected...using the expected shared secret key
- B sends challenge to A - A sends back the encrypted challenge. A sends B a challenge; B sends encrypted challenge back to A

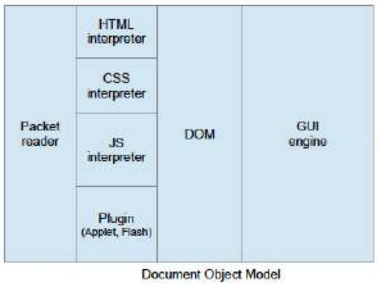
The Alive Signal

- Client wants server to know that it still wants to be connected, when its not requesting any services for server
- Protocol
  - A. Using handshaking timeout agreement. Client sends a packet saying here every x time to host
  - B. If host does not receive a ping, it sends ping request to client
  - C. If no ping sent back from client, connection is lost.
- **Connection lost = port number freed on host**

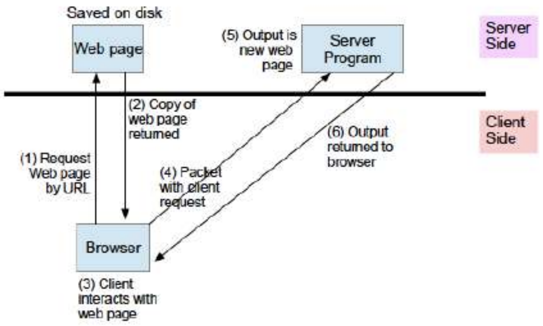
Example tools used for Stack

- Client:
  - GUI: html, css, cgi form
  - Processing: javascript
- Communication
  - Payload: cgi packet using json data
- Server side:
  - Server: apache
  - Programs: php and c
  - Content: mongo database

The Browser System



Front-end Landscape


Saved on disk
Web page
(5) Output is new web page
Server Program
Server Side
Client Side
(2) Copy of web page returned
(6) Output returned to browser
(1) Request Web page by URL
(4) Packet with client request
Browser
(3) Client interacts with web page

XAMPP

- Cross-platform support
- Server, database, PHP and Perl

Node.js

- Javascript
- For scalable network applications
- Event-driven, lightweight
- Real-time data transmission

FRONTEND

HTML = Hyper-text markup language

- Kid of SGML (standard generalized markup language)
- **Markup language**: text formating language
- Format: open and close
- Document, page and text formatting

<html> <head>setup info for page</head> <body>webpage</body></html>

- Bullet: ul, li; Number ol; Table: tr, td ; Extra space: &nbsp; Canvas: draw shit
- Deprecated: Items removed from a language...use HTML5

CSS = cascading style sheet

- 3 elements: selector, properties values
- Try not to use inline style

Order of priority

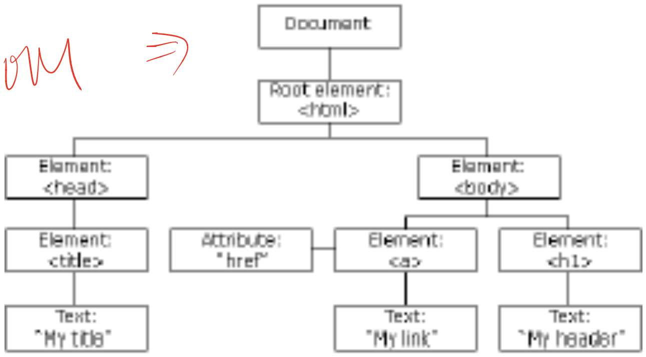
1. Browser default
2. External stylesheet
3. Internal stylesheet (declared inside head)
4. Inline style (inside element)

- If overlap, inline take priority

DOM = Document object model

- Data structure defines what should be displayed on the >> computer screen
- It is a tree: each node is an element, pointer point in the direction the elements need to be rendered on the screen

DOM ⇒



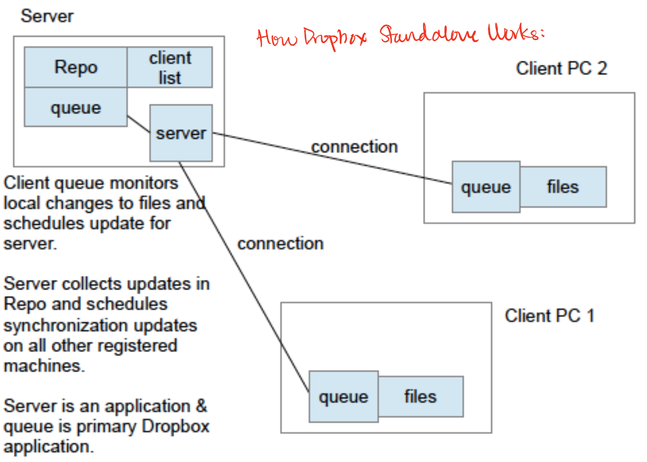
Dynamic Programming

- Since DOM is a tree, nodes can be deleted/changed, pointers can be moved
- Html create DOM nodes, css modify nodes
- JS interacts with user to modify and create node

Standalone applications

- A network-based application that does not use a browser as its primary mode of connection between client and server
  - Ex. Dropbox desktop application

How Dropbox Standalone Works:



Client queue monitors local changes to files and schedules update for server.

Server collects updates in Repo and schedules synchronization updates on all other registered machines.

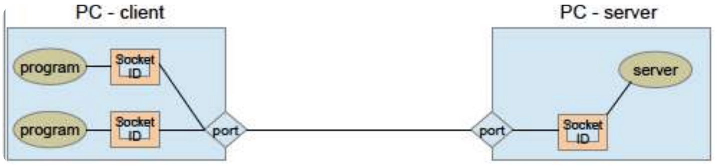
Server is an application & queue is primary Dropbox application.

Socket programming

- Send info between programs
- Port: physical; socket: logical
- Socket is the network IF of the program: IP:SID:PID

URL Connection String: Http://www.k.com:80/file

- http:// : universal string location name
- :80 port number the socket is attached to
- /file... path name

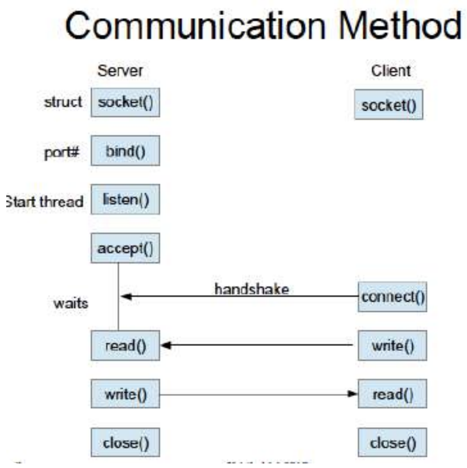


Socket

- A method by which an application program can connect with the network
- A client and server must handshake a common socket connection to communicate

Programming forms

- A.out or .exe
- Browser based client communication: server need to support browser sockets



INTERPROCESS COMMUNICATION

Load Issue

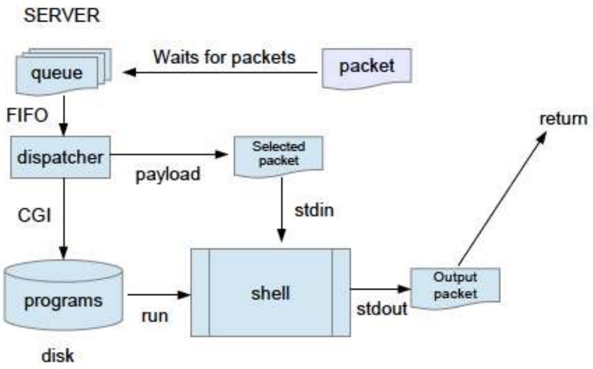
- For client to server communication, one server has many clients
- So may overload, affect response time, memory and throughput

Ways to communicate

- REST
  - Servers wait in a busy loop for a random query packet
  - If query packet is valid, perform request and return result to source address
- PUSH
  - Server stores a database of source addresses
  - At some event, send a packet to subset/all sources addresses
- PULL
  - Client has server address, and on a timer(of regular intervals) queries the server

REST Protocol = representational state transfer

- Favoured over SOAP bc does not need much bandwidth on the server
- Decouples the client from server, allowing them to run independently from one another
- **Stateless:** no longterm memory
- **Layered:** multiple technologies work together
- **Uniform interface**: one way to communicate
- Server doesn't save anything and use standard packet and CGI communication

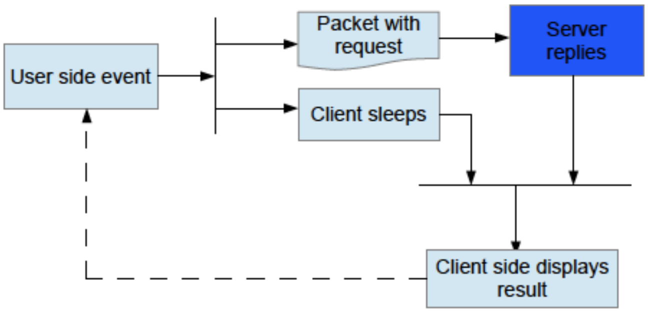


REST: Server

- Queues all requests
- One at a time, creates a session with the first item in queue
  - STDIN = incoming packet payload
  - STDOUT = outgoing packet payload (returning)
- User requests a program to run (layered). The server does not know how to process the request, assume requested program knows
- At end of program execution, outgoing packet is sent to client and session is deleted...onto next queued item

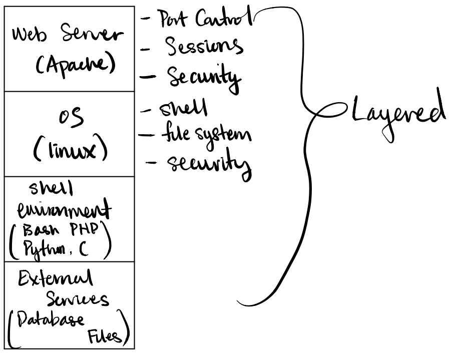
REST: client

- Application sends request to server
- Application sleeps until server returns reply
  - Ex. Browser normally will continue to let you scroll
- When reply arrives
  - Application wakes up
  - Window is refreshed with new info from the reply packet



REST Protocols

- UDP-User Datagram Protocol
  - Connection-less, delivery not guaranteed
  - REST server is not required to return an error packet and to try again
  - REST server does not return any replies
- TCP - transmission control protocol
  - Connection-based, delivery guaranteed
  - REST server is required to return an error packet after 3 attempt(default)
  - REST server must return a reply: error, success or result of request
- REST Pros: standard API, high throughput
- REST Cons: limited build in security and session history



Notice that the driving force is not the server

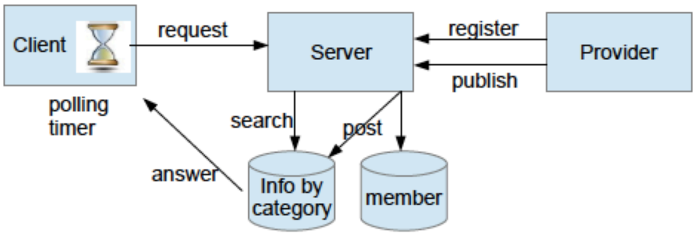
PULL

Optionally can implement with client registration

The Client server is a REST server

PUSH

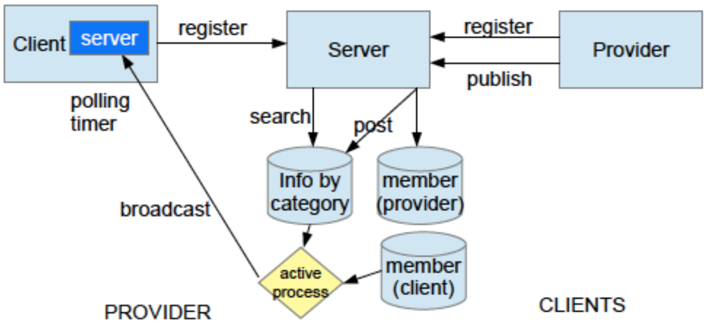
It is the POST that activates the process



CLIENT
a) Sends request using API + Topic

SERVER
a) Search for topic in DB
b) Returns result

PUBLISHER
a) First registers
b) Receives password
c) Sends updates using API



PROVIDER

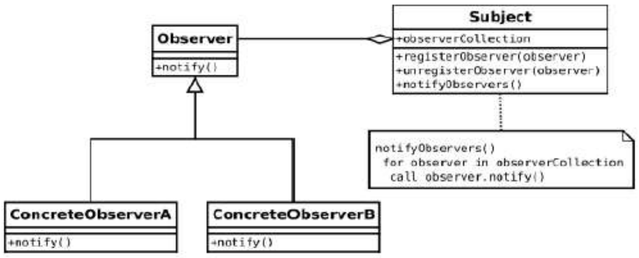
PID	TOPICS	DATA

CLIENTS

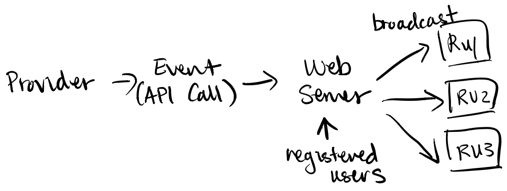
IP	TOPICS	Other

Observer Design Pattern:

1. Define an observer interface type. Observer classes must implement this interface type
2. The subject maintains a collection of observer objects
3. The subject class supplies methods for attaching observers
4. Whenever an event occurs, the subject notifies all observers.



How to implement observer:



1. Users must register
2. Random providers send msg
3. Get IP of registered users
4. Broadcast to registered users

Data Transmission

- Server --{{communication pathway as str}}--Client
- CGI format: “ http://URL/PATH/program?var=val&var2=val2”
    - URL: Internet
    - PATH & program: OS
    - Payload as value-pairs: SHELL

Different Forms of Data - How to represent in a packet?

- Streams: GPS locations (discrete, continuous)
- Transactions: bank transactions (discrete, atomic)
- Code serialization: convert objects into strings (sequential)

Server HTTP Status Codes

- HTTP is a protocol to transfer HTML pages using REST; culmination of several RFC's
- A HTTP client establishes a connection over a predefined port: 80 normal HTTP; 443 for SSL HTTP
- Server sends back a response code with the requested document, if it exists
    - 200 = OK; 401 = unauthorized; 403=forbidden; 404 = not found; 500=internal server error

CGI = common gateway interface

- Standard way to format requests; uses TCP protocol
- <form>: 2 ways to send data: post/get

CGI to packets

```
<form action="URL" method="GET">
        <textarea name="feedback">
                some text
        </textarea>
        <input type="submit" value="send" name="button">
</form>
```

Resulting payload in packet: http://URL?feedback=some+text&button=send

GET method

- Transfers data inside the query string
- Allows easy use of back button
- Easy to debug
- Less secure since text is transferred in query; data auto logged in server
- *Data placed in stdin: readable by scant, gets etc*

POST method

- Transfers data as part of the payload only
- More secure; not in query string
- Data not auto logged
- Not good with back buttons: warning that data needs to be posted again
- Harder to debug
- *Data placed into shell memory; readable by shell memory commands getenv()*

Standard Data Formats

- String = custom format
- CSV = comma separated records
- CGI, JSON = variable value pairs
- XML = tag attribute statements
- SOAP = XML based data interchange

XML

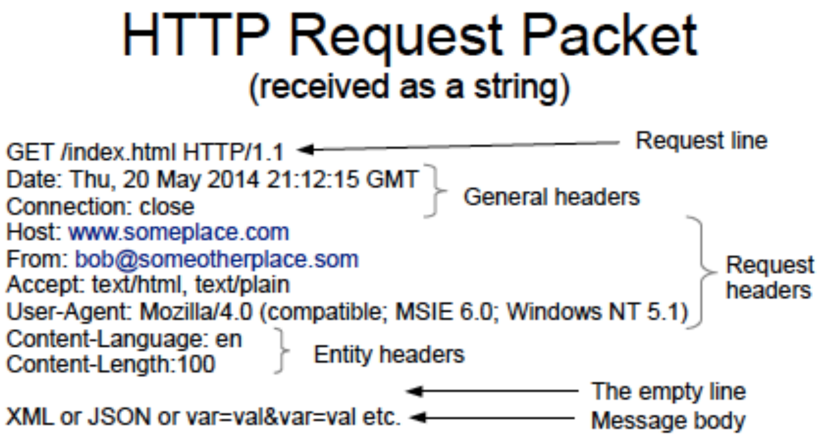
- As a communication tool
    - ○ Format with XML rather than CGI and need to replace post str with javascript conversion of msg
- As a database tool
    - ○ Webpage that doubles as a way to store records and fields of info
    - ○ Similar to css but more readable form; supports format validation
- XML formats data

XML DTD and Validation (NEED TO WATCH LECTURE)

JSON (NEED TO WATCH LECTURE)

- Communication tool that use javascript to replace CGI POST string with JSON formatted string
- Easier to process because more compact representation
- No built-in validation; just a formatting style

BACKEND - WEB SERVERS



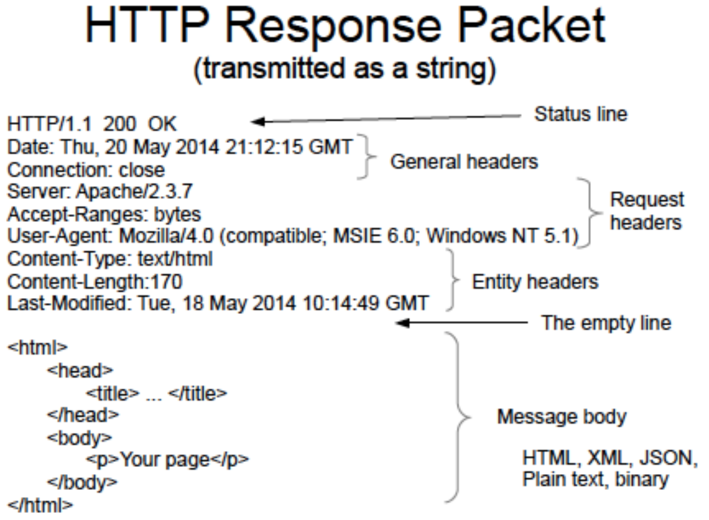
HTTP Request Packet

(received as a string)

```
GET /index.html HTTP/1.1
Date: Thu, 20 May 2014 21:12:15 GMT
Connection: close
Host: www.someplace.com
From: bob@someotherplace.som
Accept: text/html, text/plain
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Language: en
Content-Length:100

XML or JSON or var=val&var=val etc.
```

Request line
General headers
Request headers
Entity headers
The empty line
Message body



HTTP Response Packet

(transmitted as a string)

```
HTTP/1.1  200  OK
Date: Thu, 20 May 2014 21:12:15 GMT
Connection: close
Server: Apache/2.3.7
Accept-Ranges: bytes
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Type: text/html
Content-Length:170
Last-Modified: Tue, 18 May 2014 10:14:49 GMT

<html>
    <head>
        <title> ... </title>
    </head>
    <body>
        <p>Your page</p>
    </body>
</html>
```

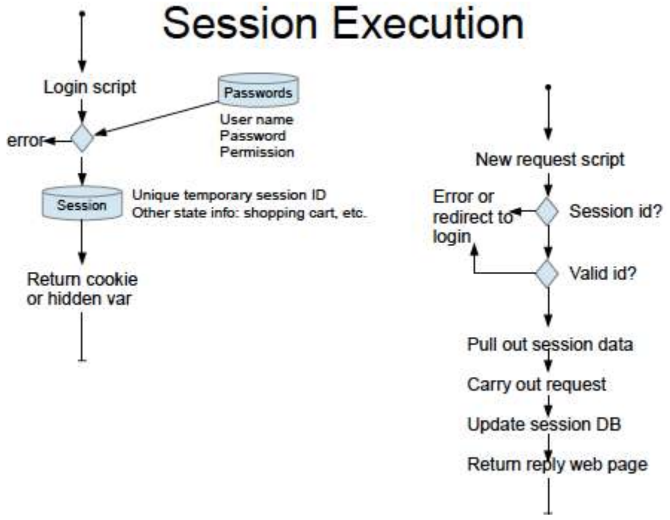
Status line
General headers
Request headers
Entity headers
The empty line
Message body
HTML, XML, JSON, Plain text, binary

Request time user = request+server+response

Request timer server = session+security+program+database

Server types

- XAMPP - session driven; cross-platform, multi-tech, OS-driven
  - Supports multiple languages and sessions
  - Table-based database for complex queries
- MEAN - even driven; single language, no-SQL
  - In java
  - Pointer-based databases for fast single interactions
- DJANGO - database driven; min programming, framework-based
  - Fast development but slowest execution platform



Session Execution

Apache

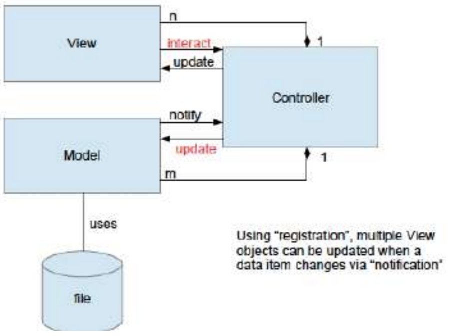
- HTDOCS: directory for files u want to serve and web-viewed
- HTTPD.CONF: HTTP server
  - Httpd.conf - main config file
- CGI-BIN: examine httpd.conf for AddHandler & ScriptAlias to find script files and dir that has CGI scripts

Session Packet

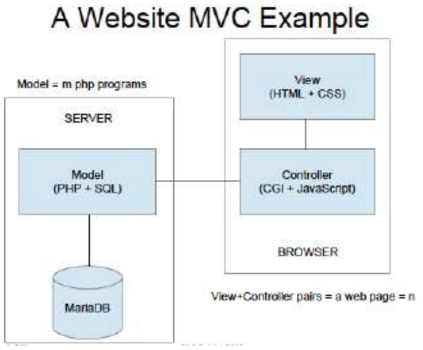
- IP+Port+SID+Ticket+CMD+Payload
- SID: each packet after login doesn't need username ad ps to validate packet; timeout defined new SID
- Ticket: a permission ID logged to DB
  - DB=username+ticket+permission+date
  - Validation by challenge response; date limit cancels ticket/logout cancels ticket

DJANGO

- MVT - Model View Template
  - controller= DJANGO
  - Template = HTML+Django template language
  - View = model + template
- MVC
  - Definition: pattern design that logically decides data drive applications into encapsulated components
  - controller = routes the request from view to model and back again
    - Submit, links, command line, menu
  - Model = stores data and is API for data
    - Database on disk; data structure in RAM
  - View = displays shit on screen
    - Table, form, diagram etc.
  - User interacts with view(sees) and controller (uses)



Using "registration", multiple View objects can be updated when a data item changes via "notification"



A Website MVC Example

View+Controller pairs = a web page = n

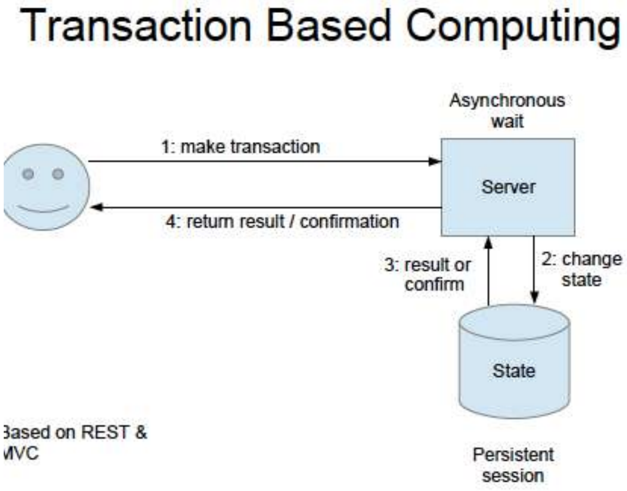
Framework

- Engine = Framework with a main execution style
  - Main method invokes callback methods in some order
  - Developer implements the callback methods and invoked by main execution cycle
- Framework = framework without a main execution cycle (ex. Bootstrap(front) and Slim for back)

- Web framework hides implementation on details of MVS and HTTP request processing

TRANSACTION BASED COMPUTING

- Based on MVS (Predominately a sever interaction model)
- Applications with the following properties:
    - Communication based on transactions
        - A query resulting in a single atomic action/change
            - eg: delete file, deposit $
    - Server could “rewind” requests to restore the server's state to a previous time
        - Assumes logged transaction
            - Log contains state before and after transaction + request
    - Security and redundancy and confirmations
- Process
    - Server waits for short duration request
    - State changes at the server
    - Reply is returned to client
    - This is independent from REST or dedicated Socket connection architectures


Transaction Based Computing

Heavy Transactions

- Single large packet, or a large segmented series of packets (ex. Entire web page)

Light Transaction

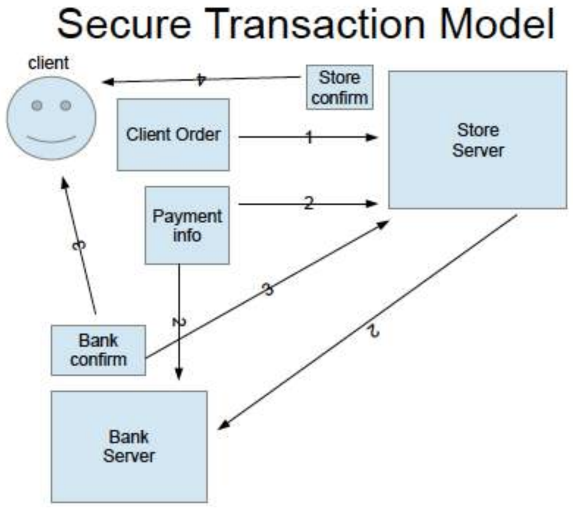
- A small single packet
- Returns a single string or database record
- Returns data, not a webpage

Architecture

- Client and server as black-box services that support:
    - URL/PATH to identify resource
    - Intermediate message state
        - String, JSON, XML, CGI
    - Using f HTTP verbs
        - PUT, GET, POST, DELETE
    - Usage of Hypertext links to resource and data

HTTP Verbs

- GET: ask for info, do not modify server info, safe server interaction
- PUT: edit/update server info
- POST: create new record of info
- DELETE: delete an existing record of info


Secure Transaction Model

Common Transactions

- Update a field in a database - optional reply
- Query a database - mandatory reply
- Ask to execute a script - scripts normally terminate quickly
- Ask for a webpage - already in HTML or generated using a program

Example transactions:

Facebook

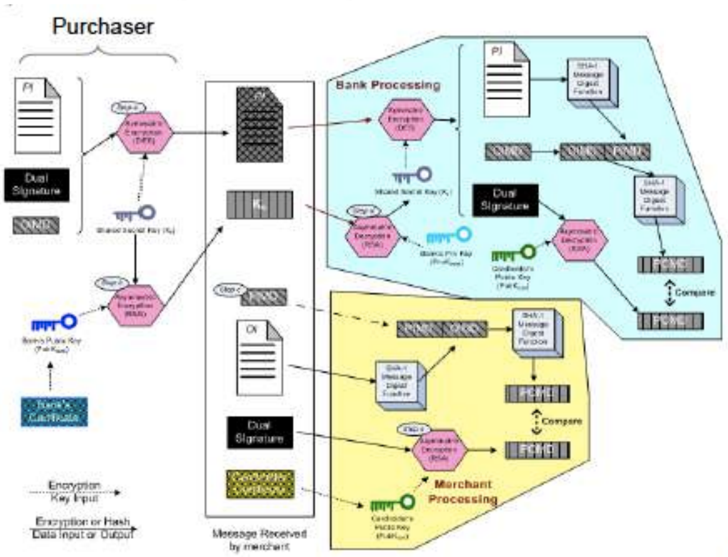
- Like (light)
- Post (standard)
- Home screen (heavy)

Amazon

- Shopping card (light to standard)
- Payment (standard)
- Search results (heavy)

Secure Transaction Based Model

- Encrypt request and result messages
- Double entry confirmation messages



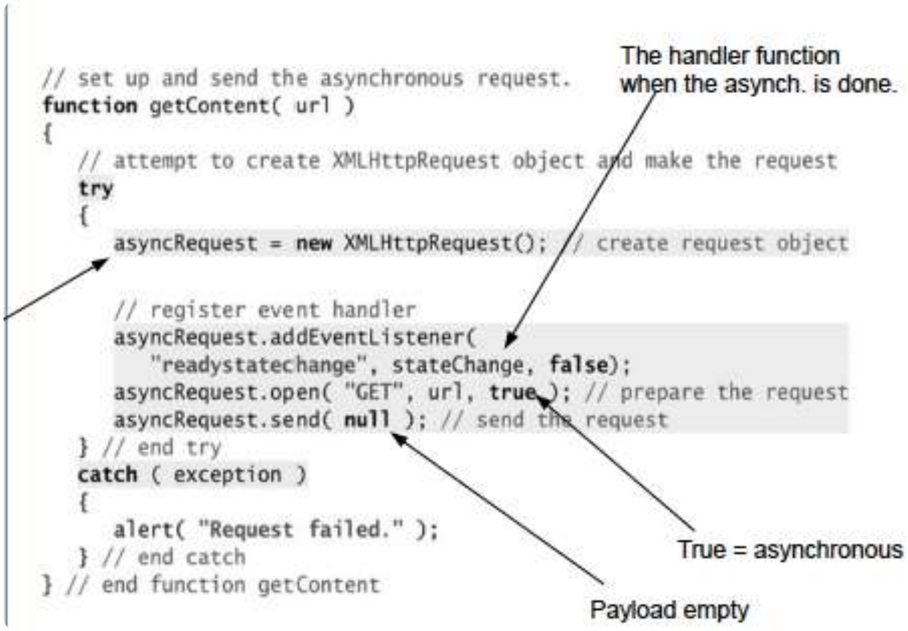
AJAX&ASYNCHRONOUS TRANSACTIONS

Rich Internet Applications(RIAs)

- web apps that approx the look, feel and usability of desktop apps.
- Two key attributes of RIAs: (1) performance and (2) a rich GUI
- Performance comes from Ajax (asynchronous javascript and XML)
    - Uses client-side scripting to make web applications more responsive
    - Ajax apps separate client-side user interaction and server communication and run them in *parallel*
- Raw Ajax uses JS to send asynchronous requests to the server, then updates the page using the DOM
- Ajax tool-kids, such as jQuery, ASP.NET Ajax and JSF's Ajax capabilities, which provide powerful ready-to-use controls and functions that enrich web applications

AJAX Application example - Form submission

- When user did not fill required fields and clicks register
- Server responds by indicating invalid fields.



```
// set up and send the asynchronous request.
function getContent( url )
{
   // attempt to create XMLHttpRequest object and make the request
   try
   {
      asyncRequest = new XMLHttpRequest(); // create request object

      // register event handler
      asyncRequest.addEventListener(
         "readystatechange", stateChange, false);
      asyncRequest.open( "GET", url, true ); // prepare the request
      asyncRequest.send( null ); // send the request
   } // end try
   catch ( exception )
   {
      alert( "Request failed." );
   } // end catch
} // end function getContent
```

The handler function when the asynch. is done.

True = asynchronous

Payload empty

- The page does not reload like a traditional app. Page was locally updated by modifying the DOM and seeing that result immediately on the screen

jQuery- makes javascript writing easier and shorter

Certificates contains a bunch of certificates

- Compare chain for trust
- X.509 Certificates

UDP Protocol

- How to communicate with protocol
- Only hop and no end (bc end is that one that sends ACKS)
- Things that require speed (email and streaming) bc ACKS are slow

TCP

- Hop and end - Everything is based on this

Handshake Protocol

- Might include cryptographic authentication inside
    - Assuming that you've registered previously
    - Keys are passed after registration
    - And see if we agree if keys match
    - So when you login, server challenges to see if u are who you are. Gives you a msg and asks you to encrypt it with the previous key given to you at registration
    - User also wants to challenge server etc.

Classic Web Application

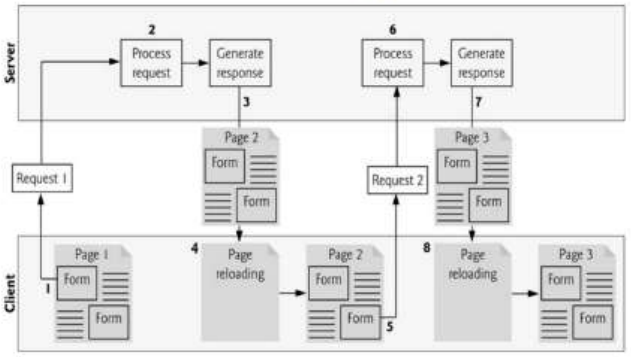


Fig. 16.1 Classic web application reloading the page for every user interaction.

AJAX Application

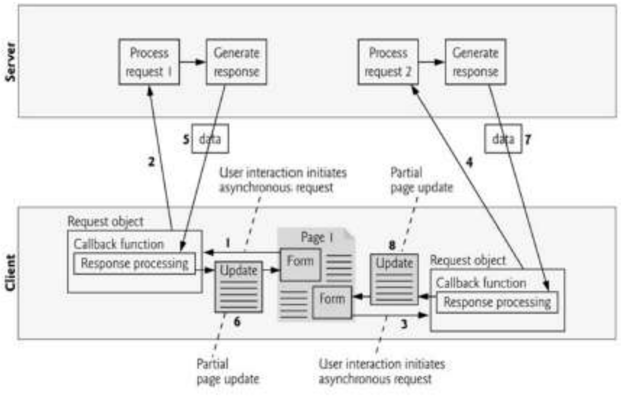
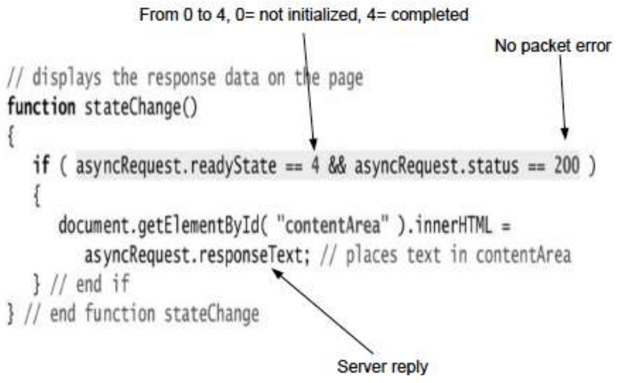


Fig. 16.2 Ajax-enabled web application interacting with the server asynchronously.



From 0 to 4, 0= not initialized, 4= completed

No packet error

```
// displays the response data on the page
function stateChange()
{
   if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
   {
      document.getElementById( "contentArea" ).innerHTML =
         asyncRequest.responseText; // places text in contentArea
   } // end if
} // end function stateChange
```

Server reply

Alive Signal

- If ur logged in long enough, bad guy can figure out key eventually
    - Change ur key every hour etc.

Same server and client but certificate authority

- Server and client gets a certificate
- Server saves private key and share public key
- Same for client
- Start communication, encrypt using public key of client or encrypt using public key using server

Sharing secret key

- Both generates key so needs common software on each side that understands each other
- someone determines a 100 digit prime number p with extra root g
- Share them publicly within a public packet
    - Server and client computes their own X and Y: g^x mod p and g^y mod p
        - Little x and little y are secret
    - Share x within unencrypted packet again
    - Take the opposite number and do k = Y^x mod p and k' = X^y mod p
    - k should equal to k'
        - K = k' = g^(xy) mod p
- CON: need algorithm/common software on both sides