CS162
Operating Systems and Systems Programming
Lecture 25

Review

April 27, 2011

Ion Stoica

http://inst.eecs.berkeley.edu/~cs162

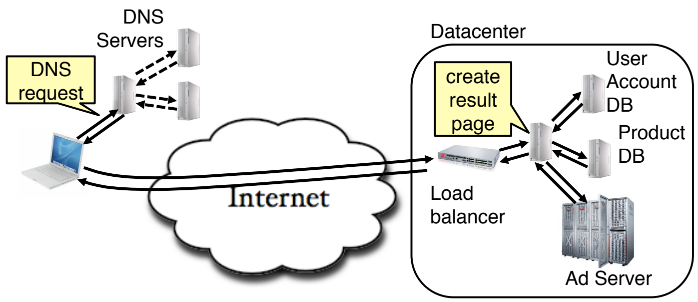
New CS162

- Gateway system class to give students a broad view on how today's systems and services
  - Better prepare students to design and develop such services
- Teach students how to develop large projects in teams
- Enable department to create a new **core** OS class (which will be offered in Spring 2012)
  - Will use a real OS for projects (likely Android)
- Enable other system classes (for which cs 162 will be prerequisite) to go deeper in their specific material and have more sophisticated projects

New vs. Old CS162

- Curriculum: 70% overlap
  - File systems, queueing theory, slightly fewer lectures on concurrency, caching, and distributed systems
  + More networking, database transactions, p2p, and cloud computing
- Different project: emphasize on how a system works end-to-end rather than focusing on implementing OS concepts in Nachos
- What if you want to do an OS project?
  - CS 163 (?) in Spring 2012
  - CS 262 graduate System class (you'll need instructor approval)
  - CS295 Cloud computing Seminar (you'll need my approval)

Example: Accessing Amazon

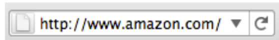


- Complex interaction of multiple components in multiple administrative domains

Universal Resource Locator (URL)

`protocol://host-name:port/directory-path/resource`

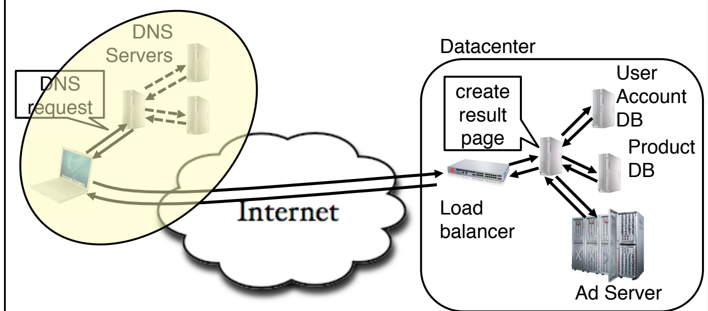
- This is what you enter in the browser!



- Example:
  http://www.amazon.com = http://www.amazon.com:80/index.html
  - `protocol` = http
  - `host-name` = www.amazon.com
    - » Name of an Amazon's web server
  - `port` = 80 (default HTTP port)
  - `directory-path` = ""
    - » Path relative to web directory at server (e.g., public_html)
  - `resource` = index.html (default file)
    - » Contains HTML home page of Amazon

Domain Name Service (DNS) Resolution

- Resolve www.amazon.com to the IP address of an Amazon HTTP server



DNS Resolution

- Resolve www.amazon.com to the IP address of an Amazon HTTP server
- How does client know DNS server
  - Client configured with the address of the local DNS server

How Does Client Communicates with DNS Server?

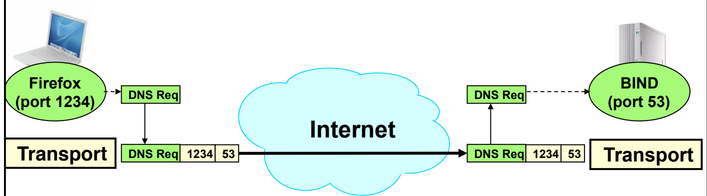
- A: Via **transport** protocol (e.g., UDP)

- Transport protocol in a nutshell:
  - Allow two application end-points to communicate
    - » Each application identified by a port number on the machine it runs
  - Multiplexes/demultiplexes packets from/to different processes using port numbers
  - Can provide reliability, flow control, congestion control

- Two main transport protocols in the Internet
  - **User datagram protocol (UDP):** just provide multiplexing/ demultiplexing, no reliability
  - **Transport Control Protocol (TCP):** provide reliability, flow control, congestion control

Transport Layer (cont'd)

- DNS server runs at a specific port number, i.e., 53
  – Most popular DNS server: BIND (Berkeley Internet Name Domain)
  – Assume client (browser) port number 1234



How does UDP packets Get to Destination?

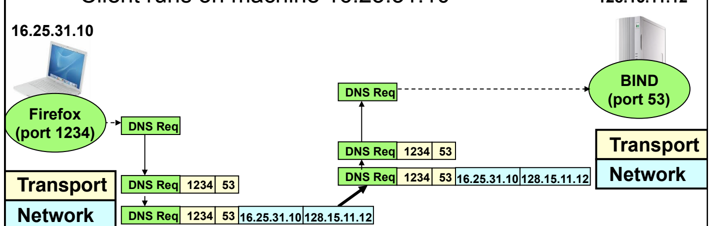
- A: Via network layer, i.e., Internet Protocol (IP)

- Implements datagram packet switching
  – Enable two end-hosts to exchange packets
    » Each end-host is identified by an IP address
    » Each packets contains destination IP address
    » **Independently** routes each packet to its destination

  – **Best effort service**
    » No deliver guarantees
    » No in-order delivery guarantees

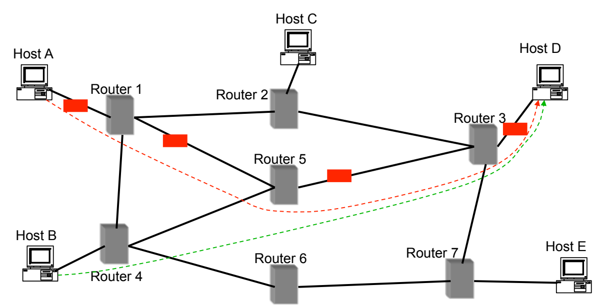
Network (IP) Layer (cont'd)

- Assume DNS server runs on machine 128.15.11.12
  – Client configured with DNS server IP address
- Client runs on machine 16.25.31.10



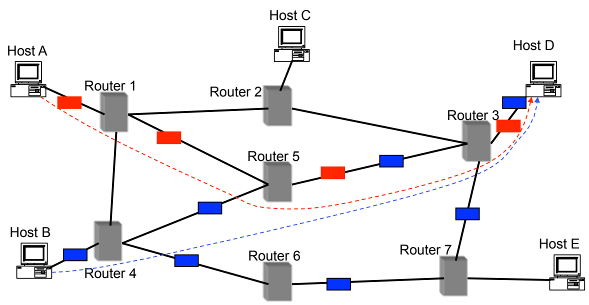
IP Packet Routing

- Each packet is individually routed



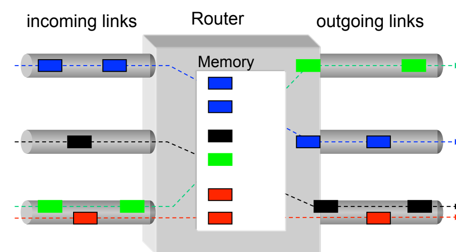
IP Packet Routing

- Each packet is individually routed



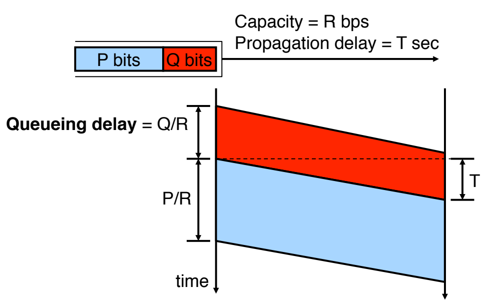
Packet Forwarding

- Packets are first stored before being forwarded
  - Why?

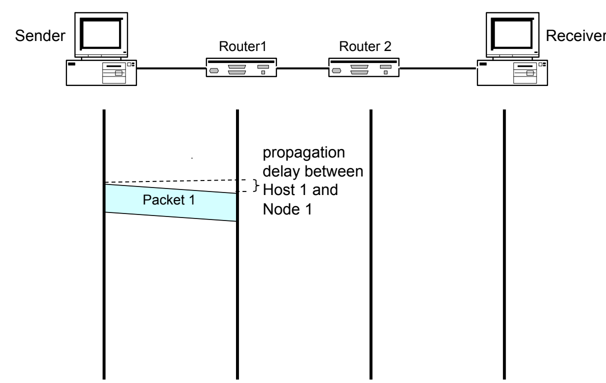


Packet Forwarding Timing

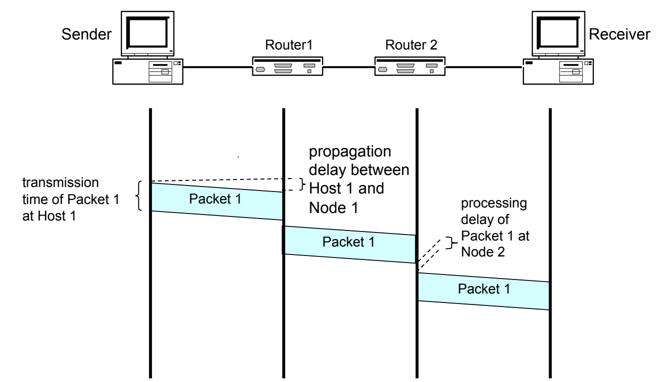
- The queue has Q bits when packet arrives → packet has to wait for the queue to drain before being transmitted



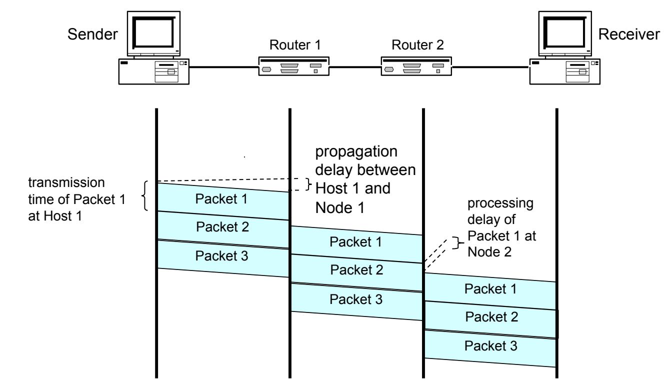
Packet Forwarding Timing



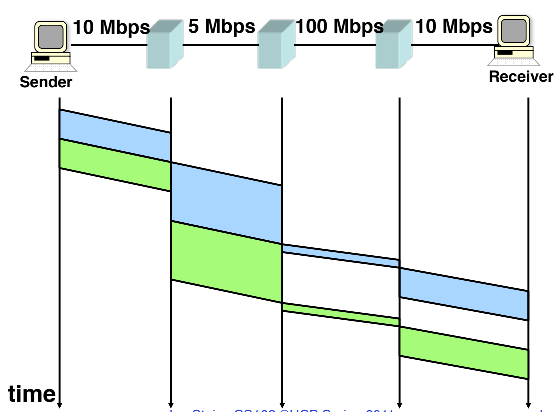
Packet Forwarding Timing



Packet Forwarding Timing

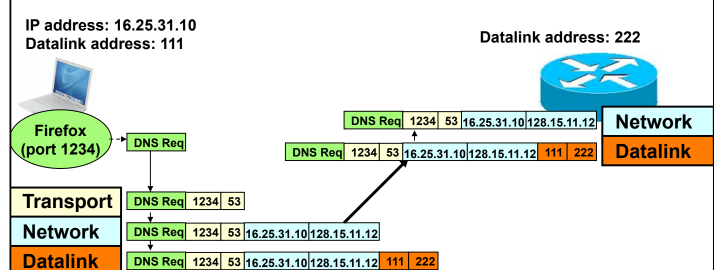


Packet Forwarding Timing: Packets of Different Lengths



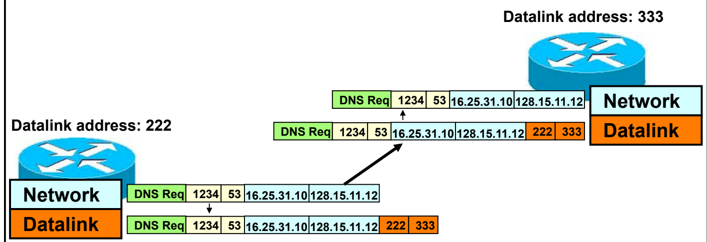
Datalink Layer

- Enable nodes (e.g., hosts, routers) connected by **same link** to exchange packets (frames) with each other
  - Every node/interface has a datalink layer address (e.g., 6 bytes)
  - No need to route packets, as each node on same link receives packets from everyone else on that link (e.g., WiFi, Ethernet)



Datalink Layer

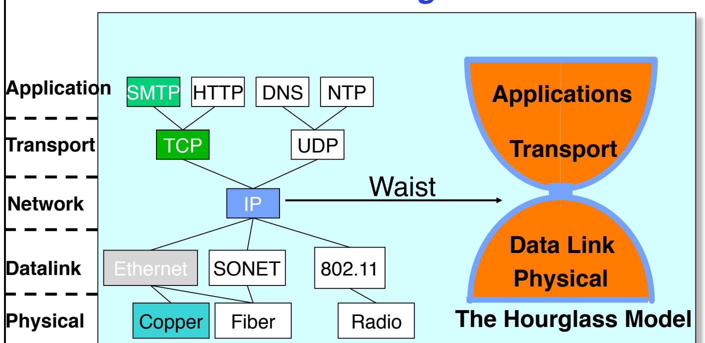
- Enable nodes (e.g., hosts, routers) connected by **same link** to exchange packets (frames) with each other
  - Every node/interface has a datalink layer address (e.g., 6 bytes)
  - **Network layer** picks the next router for the packet towards destination based on its destination IP address



Physical Layer

- Move bits of information between two systems connected by a physical link
- Specifies how bits are represented (encoded), such as voltage level, bit duration, etc
- Examples: coaxial cable, optical fiber links; transmitters, receivers

The Internet *Hourglass*



There is just one network-layer protocol, **IP**
The "narrow waist" facilitates interoperability

Implications of Hourglass & Layering

Single Internet-layer module (**IP**):

- Allows arbitrary networks to interoperate
  - Any network technology that supports IP can exchange packets
- Allows applications to function on all networks
  - Applications that can run on IP can use any network technology
- Supports simultaneous innovations above and below IP
  - But changing IP itself, i.e., **IPv6**, very involved

Application Layer: DNS Resolution

- Resolve www.amazon.com to the IP address of an Amazon HTTP server
- How does client know DNS server
  - Client configured with the address of the local DNS server

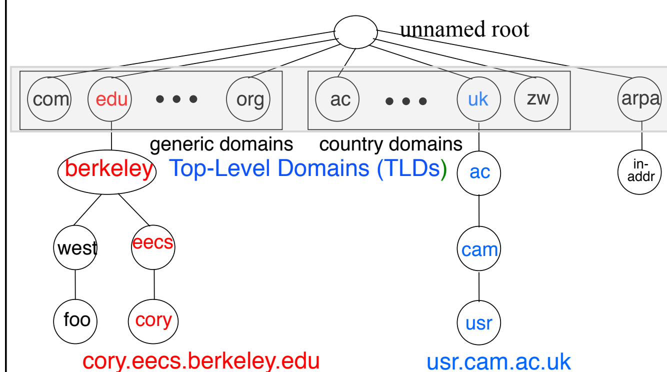
DNS: Separating Naming and Addressing

- Names are easier to remember
  - www.amazon.com vs. 72.21.211.176
- Addresses can change underneath
  - Move www.amazon.com to 76.21.211.150
  - E.g., renumbering when changing providers
- Name could map to multiple IP addresses
  - www.amazon.com to multiple replicas of the Web site
  - Enables
    - » Load-balancing
    - » Reducing latency by picking nearby servers
    - » Tailoring content based on requester's location/identity
- Multiple names for the same address
  - E.g., aliases like www.amazon.com and amazon.com

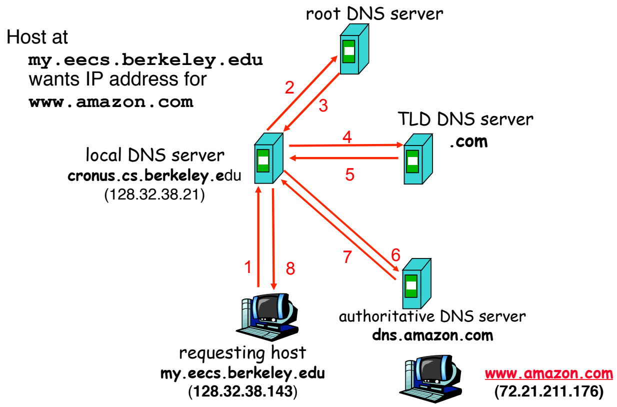
Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Zones distributed over collection of DNS servers
- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
- Performing the translations
  - Local DNS servers
  - Resolver software

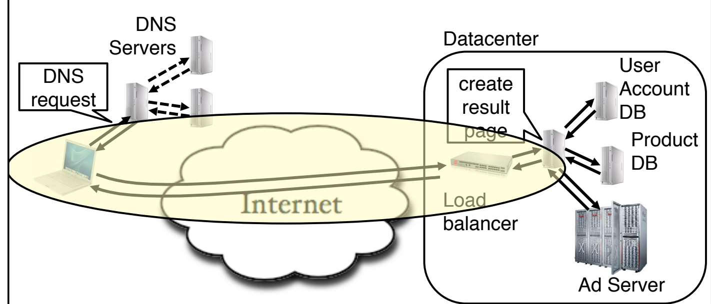
Distributed Hierarchical Database



Example

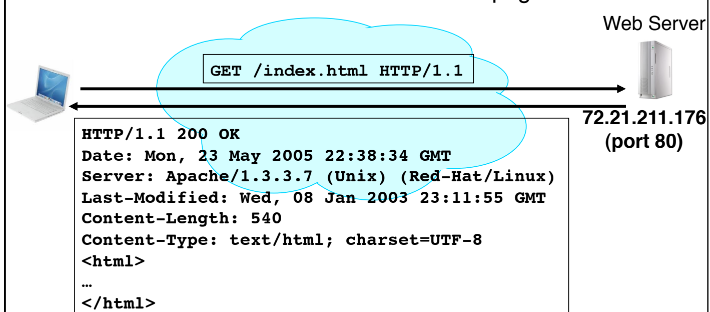


HTTP (HyperText Transport Protocol)



HTTP Request

- After resolving DNS request for www.amazon.com to 72.21.211.176 client sends an http GET request to the web server
- Web server returns HTML file for home page



HTTP Request

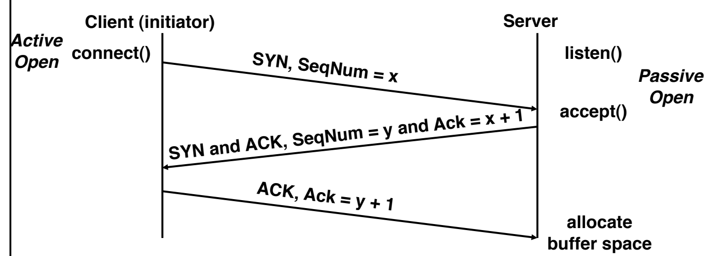
- After resolving DNS request for www.amazon.com client sends an http GET request to the web server
- Web server returns HTML file for home page
- Client renders the page
  – Need to GET other resources referred in the page

HTTP over TCP

- HTTP runs over TCP not UDP
  – Why?
- TCP: stream oriented protocol
  – Sender sends a stream of bytes, not packets (e.g., no need to tell TCP how much you send)
  – Receiver reads a stream of bytes
- Provides reliability, flow control, congestion control
  – **Flow control:** avoid the sender from overwhelming the receiver
  – **Congestion control:** avoid the sender from overwhelming the network

TCP Open Connection: 3-Way Handshaking

- Goal: agree on a set of parameters: the start sequence number for each side
  – Starting sequence numbers are random



TCP Flow Control & Reliability

- Sliding window protocol at byte (not packet) level
  – Receiver tells sender how many more bytes it can receive without overflowing its buffer (i.e., AdvertisedWindow)
- Reliability
  – The ack(nowledgement) contains sequence number N of next byte the receiver expects, i.e., receiver has received all bytes in sequence up to and including N-1
  – Go-back-N: TCP Tahoe, Reno, New Reno
  – Selective acknowledgement: TCP Sack
- We didn't learn about congestion control (two lectures in ee122)

How do You Secure your Credit Card?

- Use a secure protocol, e.g., HTTPS
- Need to ensure three properties:
  – Confidentiality: an adversary cannot snoop the traffic
  – Server authentication: make sure you indeed talk with Amazon
  – Integrity: an adversary cannot modify the message
    » Used for improving authentication performance
- Cryptography based solution:
  – General premise: there is a key, possession of which allows decoding, but without which decoding is infeasible
    » Thus, key must be kept secret and not guessable

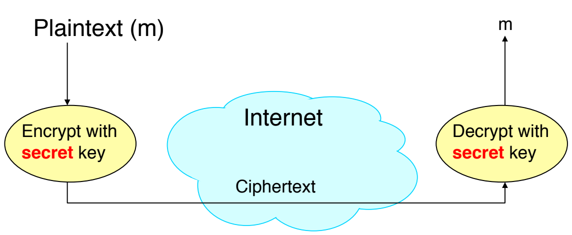
Administrivia

- Final:
  - Friday, May 13, 8-11, 2060 VLSB (this room!)
  - Closed book, **two** page of hand-written notes (both sides)
- Topics:
  - 30% first part
  - 70% second part
- Review session: Wednesday, May 5, 6-8pm, 306 Soda Hall
- Office hours:
  - Wednesday, May 4, 3-4pm
- Example questions for final already on-line
  - We'll add a few more

5min Break

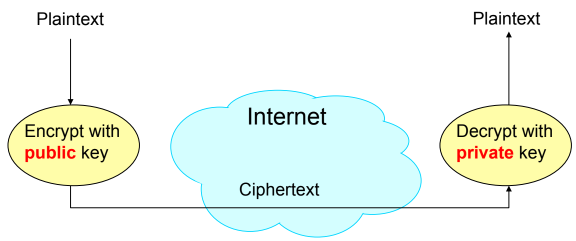
Symmetric Keys

- Sender and receiver use the same key for encryption and decryption
- Examples: AES128, DES, 3DES



Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret
- Example: RSA



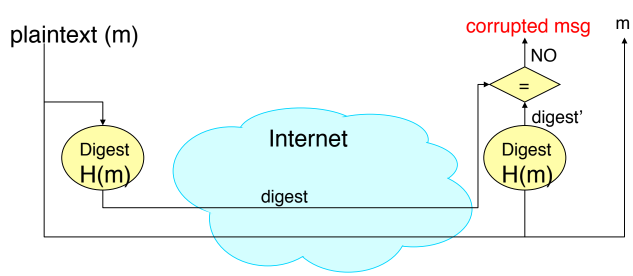
Symmetric vs. Asymmetric Cryptography

- Symmetric cryptography
  - + Low overhead, fast
  - – Need a secret channel to distribute key
- Asymmetric cryptography
  - + No need for secret channel; public key known by everyone
  - + Provable secure
  - – Slow, large keys (e.g., 1024 bytes)

Integrity

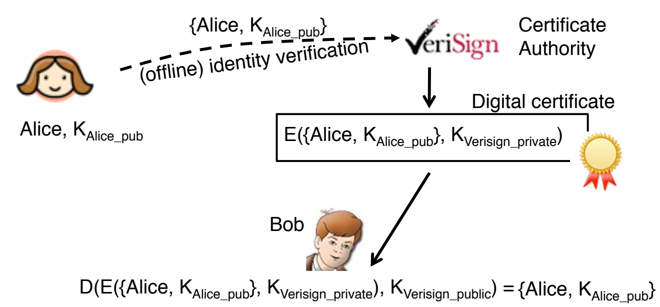
- Basic building block for integrity: *hashing*
  - – Associate hash with byte-stream, receiver verifies match
    - » Assures data hasn't been modified, either accidentally - or maliciously
- Approach:
  - Sender computes a *digest* of message m, i.e., H(m)
    - » H() is a publicly known *hash function*
  - Send digest (d = H(m)) to receiver in a secure way, e.g.,
    - » Using another physical channel
    - » Using encryption (e.g., Asymmetric Key)
  - Upon receiving m and d, receiver re-computes H(m) to see whether result agrees with d
- Examples: MD5, SHA1

Operation of Hashing for Integrity



Digital Certificates

- How do you know $\mathbf{K_{Alice\_pub}}$ is indeed **Alice**'s public key?
- Main idea: trusted authority signing binding between Alice and its private key

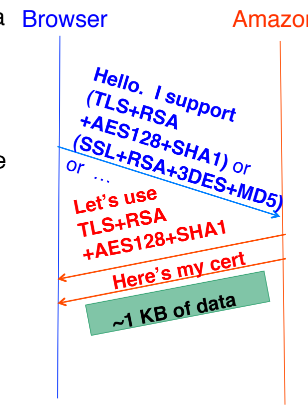


HTTPS

- What happens when you click on https://www.amazon.com?
- https = "Use HTTP over SSL/TLS"
  - SSL = Secure Socket Layer
  - TSL = Transport Layer Security
    - » Successor to SSL
  - Provides security layer (authentication, encryption) on top of TCP
    - » Fairly transparent to applications

HTTPS Connection (SSL/TLS), con't

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client sends over list of crypto protocols it supports
- Server picks protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)



Inside the Server's Certificate

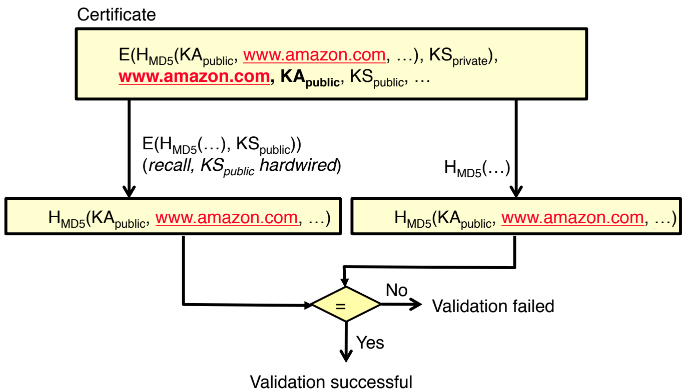
- Name associated with cert (e.g., Amazon)
- Amazon's RSA public key
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- Name of certificate's signatory (who signed it)
- A public-key signature of a hash (MD5) of all this
  - Constructed using the signatory's private RSA key, i.e.,
  - Cert = $E(H_{MD5}(KA_{public}$, www.amazon.com, …), $KS_{private})$
    - » KA_{public}: Amazon's public key
    - » $KS_{private}$: signatory (certificate authority) public key
- …

Validating Amazon's Identity

- How does the browser authenticate certifcate signatory?
  - Certificates of few certificate authorities (e.g., Verisign) are hardwired into the browser
- If it can't find the cert, then warns the user that site has not been verified
  - And may ask whether to continue
  - Note, can still proceed, just without authentication
- Browser uses public key in signatory's cert to decrypt signature
  - Compares with its own MD5 hash of Amazon's cert
- Assuming signature matches, now have high confidence it's indeed Amazon …
  - … assuming signatory is trustworthy

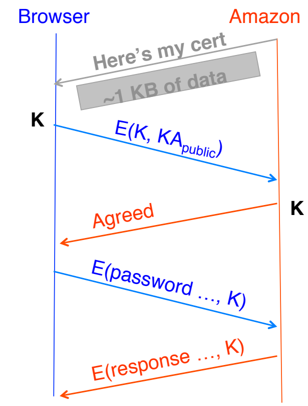
Certificate Validation

- You (browser) want to make sure that KA_{public} is indeed the public key of www.amazon.com



HTTPS Connection (SSL/TLS), con't

- Browser constructs a random *session (symmetric) key* K
- Browser encrypts K using Amazon's public key
- Browser sends $E(K, KA_{public})$ to server
- Browser displays 🔒
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) using key K
  - E.g., client can authenticate using a password



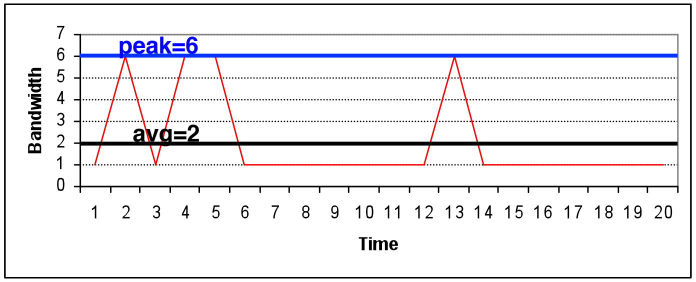
Two Key Concepts

- Statistical Multiplexing
- Name Resolution

Statistical Multiplexing

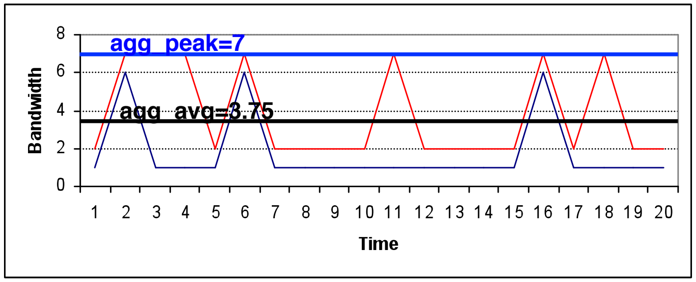
- Key to increase resource utilization
- Run multiple jobs whose peak demands exceed system capacity
  - Main idea: this is fine as long as their demands are not correlated, i.e., they don't peak at the same time!
- Widely used concept:
  - Networking: aggregate of max flow rates exceeds link capacity
  - Memory: all programs on a computer are unlikely to fit all in memory at the same time
  - Cloud services: not provisioned for every customer's workload peaking at the same time
  - Roads: not designed for all cars going in the same direction at same time
  - Banks: do not assume everyone withdraw all their money at same time
  - …

Example: One Flow



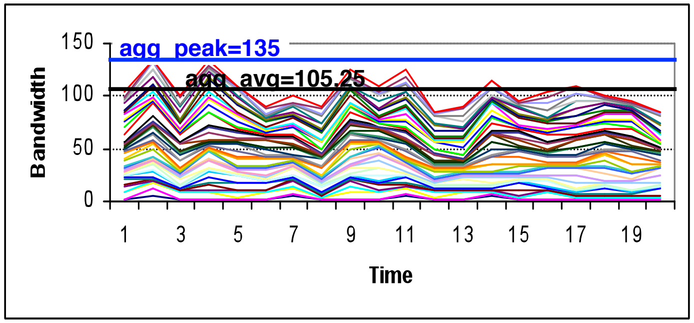
peak / avg = 3

Example: Two Flows



agg_peak / agg_avg = 7/3.75 = 1.86
(agg_avg = average of aggregate bandwidth)
(agg_peak=maximum value of aggregate bandwidth)

Example: 50 Flows



agg_peak / agg_avg = 7/3.75 = 135/105.25 = 1.28

Statistical Multiplexing (cont'd)

- As number of flows increases, agg_peak/agg_avg decreases
  - For 1000 flows, peak/avg = 2125/2009=1.057
- Q: What does this mean?
- A: Multiplexing a large enough number of flows "eliminates" burstiness
  - *Use average bandwidth to provision capacity, instead of peak bandwidth*
  - E.g., For 1000 flows
    - » Average of aggregate bandwidth = 2,000
    - » Sum of bandwidth peaks = 6,000

Lookup/Directory Services

- Resolve a name/identifier to a machine
- Name/identifier can represent
  - Machine name
  - Service name
  - Data/file name
  - …
- Challenges
  - Scale
  - Availability
  - Dynamic updates: how fast is an update propagated?

Examples: Lookup/Directory Services

- Domain Name System: map a DNS name to a server
- Service Directory: map a service to
- P2P systems
  - Napster
  - Gnutella
  - Chord

DNS Properties

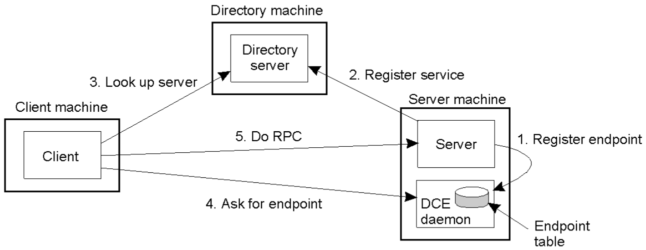
- Scale: hundreds of millions of machines
  - Hierarchy
  - Caching
- Availability:
  - Root replication
  - Caching
- Dynamic updates: slow
  - Fundamental trade-off between caching and fast updates

Service Discovery: RPC Binding

- How does client know which machine to send RPC to?
  - Need to translate name of remote service into network endpoint (e.g., host:port)
  - Binding/resolution: convert user-visible service to an endpoint
    - » Static: fixed at compile time
    - » Dynamic: performed at runtime
- Dynamic Binding
  - Most RPC systems use dynamic binding via name service
  - Why dynamic binding?
    - » Access control: check who is permitted to access service
    - » Fail-over: If server fails, use a different one

Example of RPC Binding

- Distributed Computing Environment (DCE) framework
- DCE daemon:
  – Allow local services to record their services locally
  – Resolve service name to local end-point (i.e., port)
- Directory machine: resolve service name to DCE daemon (host:port') on machine running the service

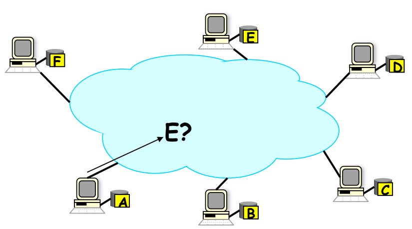


Properties

- Scale: tens to thousands
  – Single directory server "good enough" for most cases
- Availability: high, using packup
  – Backup directory service
    » Stand-by: has same state as primary directory service
    » Cold: reconstruct the state in case of failure
- Dynamic updates: fast

Peer-to-Peer Systems

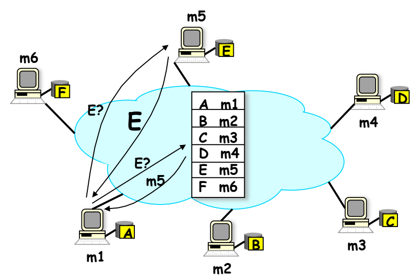
- Files/songs/videos stored across peers
- Problem: given a name or ID find the machine storing a copy of the file/vide/song with that name/ID



Napster

- Assume a centralized index system that maps files (songs) to machines that are alive
- How to find a file (song)
  – Query the index system → return a machine that stores the required file
    » Ideally this is the closest/least-loaded machine
  – ftp the file

Napster: Example



Napster Properties

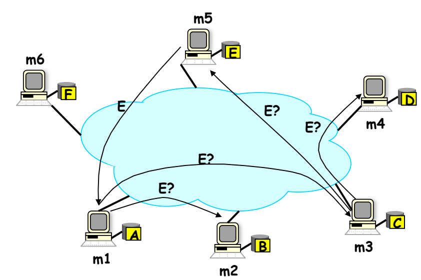
- Scalability: medimum (tens of thousands of machines)
  - Centralized directory "good enough"
  - May need to partition/replicate directory server for higher scalability
- Lookup: very fast
- Availability: high, using backup
  - Backup directory server
- Dynamic updates: fast
  - Once directory server learns about an update in the system (e.g., node leaving, joining, new file being created, deleted) every other node in the system will be aware of update

Gnutella

- Distribute file location
- Idea: broadcast the request
- How to find a file? Flood
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machine that has the file receives the request, and it sends back the answer

Gnutella: Example

- Assume: m1's neighbors are m2 and m3; m3's neighbors are m4 and m5;…



Gnutella Properties

- Scale: hard to scale to large networks due to flooding
  - To alleviate this problem, each request has a TTL
- Lookup: slow
  - Flooding network can slow everyone down
  - With TTL does not guarantee than an existing file is found
- Availability: very high
  - As long as nodes remain connected any number of nodes can fail
- Dynamic updates: very fast
  - Updates are not propagated; need only to be done locally (e.g., a new file being created or deleted)

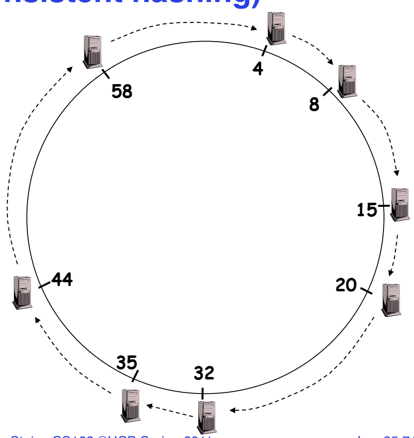
Chord Lookup Service

- Associate to each node and item a unique *id/key* in an *uni*-dimensional space $0..2^m-1$
  - Partition this space across N machines
  - Each id is mapped to the node with the smallest largest ID (consistent hashing)
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Identifier to Node Mapping Example (Consistent hashing)

- Node 8 maps [5,8]
- Node 15 maps [9,15]
- Node 20 maps [16, 20]
- …
- Node 4 maps [59, 4]

- Each node maintains a pointer to its successor

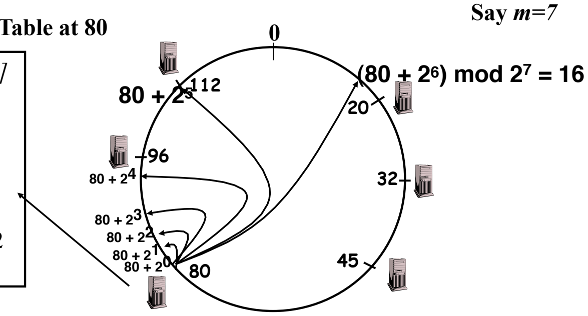


Achieving Efficiency: *finger tables*

Finger Table at 80

i	*ft[i]*
0	96
1	96
2	96
3	96
4	96
5	112
6	20

Say *m=7*



*i*th entry at peer with id *n* is first peer with id >= $n + 2^i \pmod{2^m}$

Properties

- Scale: high (tens to hundreds of thousands machines)
  - Each node needs to know about O(log N) nodes
  - Lookup takes O(log N) messages
- Lookup: fast
  - log(N) hops
- Availability: high
  - If each node maintains O(log N) successors, ring can survive with high probability to half of nodes *independently* failing
- Dynamic updates: fast
  - No caching

Not Cover in This Review

- Nothing before midterm
- Networking
  - Reliability
  - Flow control
  - E2E argument
- Database
- Most of RPC
- Chord protocol
- More on May 5, 6-8pm, 306 Soda Hall