Administrivia/Last Time

- End of the Semester
  - Still working on room for Final 5/6 at 4pm – will be posted on class website when I get it
  - 5/12 deadline for term paper and project (hard deadline if you want a grade)
  - Please email TA to set up grading time
    * Any project questions?

- Last Time
  - Naming

- Today: finish naming, p2p and maybe intro to security
Domain Name System (DNS)

- Scalable directory services protocol for the Internet
- Berkeley Internet Name Domain (bind) on unix machines
- Most common use: directory service to map from host name to IP address
Challenges and Concerns

• Challenges
  – How to build a directory system for the whole internet?
  – Can you suggest some approaches (dumb or otherwise)?
    ∗ HOSTS.txt – until mid-1980s
  – Is there a phone directory for the whole world?

• Concerns/Requirements
  – Ease of administration
  – Availability
  – Scalability
  – Security
  – Extensibility
• Idea: use a hierarchy of domains; hosts defined within a domain

• Break the domains into zones
  – Each zone administered independently

• Each zone supported by two or more name servers (why?)
  – Primary and secondary are exact copies of each other
  – Secondary polls primary for updates
Name Server

- Each server maintains a collection of resource records (RR)

- Each record: (Name, Value, Type, Class, TTL)
  - Record indicates binding Name to Value
  - Type specifies the type of binding
  - Class allows other entities to define types
  - TTL: how long the record is valid for
MX Example

- When you send mail to nael@opal.cs.binghamton.edu
- Mail program queries DNS for an MX record for opal
- The following info is returned (I used nslookup, querytype=mx):

  opal.cs.binghamton.edu  canonical name = cs.binghamton.edu
  cs.binghamton.edu     preference = 0, mail exchanger = cs.bing.
  cs.binghamton.edu     internet address = 128.226.123.101
  cs.binghamton.edu     name server=bingnet1.cc.binghamton.edu

- These correspond to:
  - (opal.cs.binghamton.edu,cs.binghamton.edu,CNAME,IN)
  - (cs.binghamton.edu,cs.binghamton.edu,MX,IN)
  - (cs.binghamton.edu,128.226.123.101,A,IN)
  - (cs.binghamton.edu,bingnet1...,NS,IN)
Name Resolution – Server Hierarchy

- Each zone managed by its own name server
- Should the name server include the full directory?
- Server hierarchy provides scalability and distributed management
Server Hierarchy

- At the top level is the root domain managed by a set of root name servers
  - Give a starting point to the full DNS database
  - Thirteen servers distributed all over the world; why not just two?
  - Manages top level domains such as .edu, .com, .net and .org
  - Also manages geographical (country) domains such as .us, .in, and even .jo
  - US domains are maintained by the Network Information Center (InterNIC)
- Next level is middle-level domains like binghamton.edu and yahoo.com
- Where do the root name servers get their information?
Name Resolution

- Site-wide cache to speed up resolution of frequently used names
- `gethostbyname()` and `gethostbyaddr()`
Caching and Replication

- 13 root servers (top level servers)
  - Last year, there was a denial of service attack on those
  - What would happen if the attack was successful?

- Caching: to reduce DNS traffic, each resolution is cached locally
  - Recommended TTL for hosts is 2 days
  - Record specifies TTL field (can set it to 0 if no caching is desired)
Who are you – reverse DNS

- the domain in-addr.arpa provides reverse mapping
- Used by servers to figure out who is connecting to them
- Records of type PTR
- nslookup with querytype=ptr on 128.226.123.101

  101.123.226.128.in-addr.arpa name = cs.binghamton.edu
  123.226.128.in-addr.arpa nameserver = bingnet1.cc.binghamton.edu
  ...

- Dynamic DNS
Aside: DNS Cache Poisoning

- Spoofing attack (security next time)
- Mallory (man-in-the-middle) asks its DNS server for the address of hotmail.com
- DNS server recursively sends the request to dns.hotmail.com
- Before it can answer, Mallory spoofs the answer telling the DNS server that hotmail.com is his own machine
- Users now connect to Mallory instead of hotmail.com and volunteer their passwords
- Solution: authenticate DNS replies/updates
Getting back to Naming

- DNS provides FQDN to IP name resolution; is this the only type of naming of interest?

- Service Discovery/Intentional Naming: equivalent to yellow pages
  - Find a service – typically not individualized
  - e.g., a printer, or a webserver ...
  - How would you implement it?
  - Would DNS work?
Naming in Peer-to-peer (P2P) Systems

- Networks made up of peers that join and leave arbitrarily and interact with other peers

- Fundamentally different model for applications (compared to client server)

- Peer-to-peer applications can provide
  - redundant storage
  - Permenance (your stuff available when you are disconnected)
  - anycast
  - anonymity
  - etc...
Core Operation of P2P

- Core operation in p2p applications: finding data
  - Given a key, find the value corresponding to it
  - This is a naming problem; why not use DNS?

- Why not DNS for P2P?
  - Hosts can come and go
  - No host hierarchy
  - No naming structure
  - Data is of interest, not the peers (machines)
Some Peer to Peer Applications

- Cooperative Mirroring: content providers cache each other’s data – provide load balancing (plan for average rather than peak)

- Time-shared storage: if intermittently connected, someone else can make your “stuff” available. When you are connected, you make theirs available in return

- Distributed index: finding data based on keyword input as in Napster/Gnutella

- Embarrassingly parallel applications (e.g., breaking code, SETI@home)
Aside: SETI@home

- Largest and best known peer-to-peer computing project
- SETI: Search for Extraterrestrial Intelligence
  - Gobs of data from radio telescopes need to be analyzed
  - No one will fund them, cant buy computing resources
- Big success:
  - Around 4 mil. clients downloaded
  - Total CPU time: 1246848 years! 54.8 TFLOPS
  - ASCI RED (biggest supercomputing cluster) is 12 TFLOPS with cost $100 million
  - Get your own copy: http://setiathome.ssl.berkeley.edu/
Naming/Search for Peer-2-Peer

- Napster – centralized index

- Gnutella – flood the “overlay network”, get responses

- Neither is scalable, especially given the resolution and update frequency

- Enter Chord (and other) research protocols in this area
  - How do we carry out such generalized resolution in such a dynamic network?
Some P2P Classification

- Structured vs. Unstructured
  - Unstructured: Data can go anywhere
    * But how to find it?
  - Structured: Data goes to a specific place
    * Makes it easier to find....but how?

- Centralized vs. Distributed
Chord Protocol

- Key idea:
  - Arrange the nodes in a logical circle, where every node knows its successor node
    * Pass the request along the circle until someone replies
    * Very slow: $O(N)$ messages needed, sequential resolution

```cpp
// ask node n to find the successor of id
n.find_successor(id)
if(id ∈ (n, successor))
  return successor;
else
  // forward the query around the circle
  return successor.find_successor(id);
```
Main Improvement – Use Routing Pointers

- Use consistent hashing to map the key to a node
  * Each node is responsible for the keys that hash to the range \([\text{predecessor}, \text{id})\)
- Keep track of your power-of-2 successors in a routing table
  * \(\log N\) entries per node; \(O(\log N)\) messages needed to resolve
- Joins/Stabilization in the background to keep pointers current
What are the issues in P2P computing?

- From the Chord review, some issues include how to join the system, how to find stuff (naming), how to route, performance.

- Most commonly, p2p is implemented as a distributed overlay system:
  - software running on top of the network that self configures and defines its own topology
  - Similar to your project
  - Usually would like it to be geographically sensitive (two nodes that are close in the overlay should be close in network distance) – why?
Peer to Peer Hot Research Area

- Many new applications; a lot of work still to be done
- Example: Freenet
  - Adaptive p2p network that allows publication and retrieval of data while protecting anonymity of publisher and reader
  - Widely used (over 2 million downloads), especially in China to get around censorship and prosecution
  - Like Chord, doesn't use a centralized index or flooding; distributed hashing to store files in a location independent way
  - Adaptively replicates file close to where they are used, and deletes copies that are not being used
Bit Torrent, E-donkey, E-mule

- Access files off-of a website
- File is chunked into pieces (256Kbytes typical)
- A tracker keeps track of who are downloading files
- New nodes downloading file get a random subset of the downloading nodes, they use them to get the file parts
- Load is distributed
- Server gets benefit of offering service (e.g., advertising), without full cost of serving the content
Other Examples

- distributed.net
  - Allows users to share their free CPU cycles
  - Instead of tracking data, have to track resources

- Grid Systems
  - Overlay network to make Many resources at member sites appear as a single supercomputer
  - P2P ideas heavily used between member sites (e.g., to map applications to nearest resources)

- Many many more...
Next Topic – Security

• What does security mean to you (not necessarily in networking terms)?

• What provides security at your home?

• How big of a deal is security on the Internet?
Security Threats

- Threats to a computer system
  - Leakage: someone manages to get access to unauthorized information
  - Tampering: someone changes programs or data
  - Resource stealing
  - Vandalism

- Threats in a distributed system
  - Eavesdropping: reading messages without authorization
  - Masquerading: sending/receiving messages using the identity of others
  - Message tampering: altering message content while it is on the way
  - Replaying: storing messages and replaying them at a later time
Security Requirements

- Privacy: Need a secure communication channel protects against eavesdropping
- Authentication: need authentication protocols for mutually suspicious clients and servers
- Integrity: a way to make sure the message was not modified
- Nonrepudiation: a signed message cannot be changed and the signatory cannot claim he did not sign it
- Need to make sure the information is fresh (replay attacks)
- Other?
  - We will talk about firewalls; a form of access control
  - If we have time, we will look at some more
    * Using bugs in servers to obtain illegal access (YABOBs, the many Microsoft bugs)
    * Denial of service
    * Viruses/worms
Our Players

- Alice and Bob are in love and want to exchange letters
- Eve wants to eavesdrop
- Mallory is a man in the middle that intercepts letters and rewrites them or writes false letters from scratch
Major tool in security toolbox – Cryptographic Algorithms

- Idea: encrypt the data before sending it
  - If Eve listens to the transmission, she cannot understand
- Ideally, encryption algorithm:
  - Easy to encrypt/decrypt for authorized receiver (with key)
  - Computationally infeasible for anyone else
- Two approaches
  - Symmetric (secret key): a single key is used for encryption and decryption
    - Key is exchanged securely out-of-band
  - Asymmetric (public key/private key)
Data Encryption Standard (DES)

- Secret Key algorithm; widely used
- Uses a 64-bit key (56 usable bits) to encrypt a 64-bit message
- Security by diffusion/confusion
Breaking DES

• Immunity to brute force attacks is a function of the size of the search

• Breaking DES
  – 1977: Diffie and Hellman, $20 mil machine, in 10 hours
  – 1987: Diffie and Hellman, $200 K machine, 10 hours
  – 1993: Wiener designed a key search chip, 5760 chips, $10 per chip, 1.5 days
  – 1997: DES challenges I, 96 days
  – 1998: DES challenges II-1, distributed.net 41 days
  – 1998: DES challenges II-2, Electronic Frontier Foundation, 56 hours
  – 1999: DES challenges III, distributed.net, 22 hours
  – Chinese lottery, 1 billion chips, less than 60 seconds

• Need a bigger key!
Improving DES

- Would like to remain compatible with all the DES hardware already out there
- Double DES. Encrypt twice with two different keys
  \[ E_k(x) = E_{K2}(E_{K1}(x)) \]
- Problem! Meet-in-the-middle attack (Merkle and Hellman)
- Triple DES: \[ E_k(x) = E_{K3}(D_{K2}(E_{K1}(x))) \]
  - If \( K1=K2=K3 \) we get backward compatibility with 56-bit DES
- Other secret key algorithms out there like Blowfish and IDEA
- Realistic to expect the key to be known?
Diffie-Hellman Key Exchange

- Problem: how to construct a private key over a public channel?
- Based on the difficulty of computing discrete logarithms of large numbers
- Needs 2 large numbers, one prime (P) and one (Q) a primitive root of P
- Alice and Bob exchange P and Q publicly
- Alice computes $A^* = Q^A \mod(P)$ for secret A
- Bob computes $B^* = Q^B \mod(P)$
- Alice and Bob exchange $A^*$ and $B^*$
- The secret key is computed by each:
  - $DHS = (B^*)^A \mod(P)$
  - $DHS = (A^*)^B \mod(P)$
- susceptible to man in the middle attack; need authentication
Public Key Private Key

- Idea:
  - Everyone has a public key that is advertised and available for all
  - Alice communicates with Bob. She encrypts the message with Bob’s public key
  - Only Bob is able to decrypt the message with his private key

- Why is this better than Secret key encryption?

- What are the requirements on the encryption algorithm?

- Idea is due to Diffie and Hellman. Often the credit is given to Rivest Shamir and Adelman (RSA) who found an implementation
• Diffie Hellman’s Scheme – Find an Encryption algorithm E, and a Decryption algorithm D such that:
  – $D(E(P)) = P$
  – Impossible to guess D from E
  – E cannot be broken by a plaintext attack

• If such E and D can be found, then we can safely make E public

• Rivest Shamir and Adleman (RSA) proposed a method for finding D and E in 1978
  – Based on the difficulty of factoring large numbers
  – Needs two large numbers ($100^{100}$ and bigger)
RSA

- Generate public key/private key pairs using theory of large prime numbers

- Algorithm:
  - Choose two large prime numbers p and q
    * define $n = p \cdot q$
    * define $r = (p-1) \cdot (q-1)$
  - Choose an encryption key e, where e and r are relatively prime
    * Two numbers are relatively prime if they have no common factors
RSA (cont’d)

- Pick d such that
  \[ d.e = 1 \mod(r) \]

- Public key is \(<e, n>\) and private key is \(<d, n>\)

- Encryption: \(c = m^e \mod n\)

- Decryption: \(m = c^d \mod n\)

- For all m smaller than n, it can be shown that E and D are inverses

- RSA is much slower than DES (2+ orders of magnitude)
Trivial Example

• Pick primes 7 and 3. \( n = 21 \)

• \( r = (p-1)(q-1) = 12 \)

• Pick e relatively prime to r; e.g., 5

• \( d \cdot 5 = 1 \mod(12) \)

• \( d = 5 \) would work

• Suppose we want to encrypt the number 4
  – \( c = 4^e \mod(n) = 1024 \mod(21) = 16 \)
  – To decrypt: \( m = c^d \mod(n) = 1048576 \mod(21) = 4 \)
More Complicated Example

- $p=907$, $q=787$; $n = 713809$
- $r=712116$; pick $e=5$ (note that 3 is not r.p. to $r$)
- $d=569693$
- Message $M = 48879$ (0xBEEF)
- $c = M^e \mod(n) = 635750$
- Decrypt: $m = c^d \mod(n) = 48879$
RSA Discussion

- $e$ and $n$ are public
- If we can factor $n$, we can find $p$ and $q$ (and from them $d$)
  - Why is this secure?
- Factoring very large numbers has been tried for centuries – very difficult problem
- RSA claimed (in 1977) that it would take 4 billion years to factor a 200 bit number (assuming 1 $\mu$ sec instruction time)
- RSA challenge: crack an RSA encrypted 430 bit message
  - Broken in 1994 by Atkins and colleagues (in 8 months, using massive parallelism and differential cryptoanalysis)
  - Important to pick good keys (Netscape vulnerability)
Other Algorithms Possible?

- Merkle-Hellman proposed using the Knapsack problem (known to be NP Complete) to generate key pairs
- Breaking the code is equivalent to finding the optimal solution to Knapsack
- Choosing a large instance of the problem, it becomes computationally infeasible
- Offered $100 for the first person to break it
- Shamir (of RSA) broke it almost immediately
- Improved the algorithm and offered $1000
- Rivest (of RSA) broke it
- Patched again (but no $10,000 prize for poor Adleman :-)
- Not widely used because of its history
Discussion

- Public key is convenient because there is no need to distributed keys
- DES is two orders of magnitude faster than RSA
- Combination?
  - Exchange secret key (called session key) using RSA. Use DES after that
  - This is the idea behind many secure protocols (PGP, ssl, ssh)
One Way Hash Functions

- Create a cryptographic checksum

- Have properties of both hash functions and one-way (encryption) functions
  - Maps strings to numeric values
  - Collisions are “rare”
  - Very difficult to find two strings with the same hash
  - Very difficult to invert

- Example MD5
Message Integrity

- One way hash functions are used to provide guarantees of message integrity
- If the message is modified, the hash will be different
- But the attacker can simply modify the hash as well!
- Two solutions
  - Hash is encrypted using the secret key
  - Provide the hash for the concatenated message and the key
- So, now we can tell if the message has been modified; what prevents an attacker from repeating a message? (replay attack)
Replay Attacks

• We are simply repeating a valid message
  – Is this dangerous?

• Very easy to prevent
  – Add a timestamp to the message – if too old, then its a replay attack
  – More generally, add *nonces*: parameters that vary with time
    * Example, sequence number
    * If the nonce is not the expected one – replay message
    * MD5 can be used to assure the nonce does not get modified
Authentication

- Prove that you are who you claim to be
- Need two pieces of information
  - Credentials (name)
  - Verifier (picture)
- Another example is name/password
  - How to guarantee the confidentiality of the passwords?
- We went through the unix password example last time
Authentication (cont’d)

• Only the two sides should know the “verifier”
  – If the password is public knowledge, it is not much use

• Idea: encrypt the password

• What are the ideal properties of an encryption algorithm?
  – Efficient to encrypt and decrypt: $E(x)$ and $D(y)$ are simple
  – Computationally infeasible to crack the code: $E^{-1}(x)$ is very difficult
  – Only the authorized receiver may decrypt
Example: Unix Passwords

- Unix passwords
  - Uses a “crypt” function based on DES
  - Stores the encrypted password (EP) in a file /etc/passwd
    * A line for each user holding EP and other information
  - User types in name and password
    * If crypt(password) = EP, the authentication is successful

- The password file is world readable, and the crypt() function is well known

- Is this scheme vulnerable?
Brute Force/Dictionary Attacks

- Crack the password
  - Try all combinations – hope to get lucky
- Dictionary attack: try words in dictionary
- Protecting against dictionary attacks
  - Add “salt” to the password so that it does not look like a dictionary word
    * Unix salt is a 12-bit number, stored in the passwd file
    * EP = crypt(strcat(password),salt)
    * Forces cracker to repeat attack 4096 times
      - Done! Feldmeier and Karn produced 732,000 most used passwords with each of the 4096 salt values. Covers 30% of passwords
  - Add delay between failed tries
    * Does not help if you have a copy of the encrypted password
    * Hide the /etc/passwd file (/etc/shadow)
  - Pick a good password!!
Sniffer Attack

- Shoulder surfing: someone peeking over your shoulder sees your password

- Remote login:
  - You telnet to a machine, and type name/password
  - The name/password are sent in plaintext (un-encrypted) to the remote machine
  - Someone sniffing the link can learn the password

- Need confidentiality to have effective authentication
  - Need the encryption to be end-to-end
Authentication: Simple 3-way Handshake

Where do the handshake keys come from?
- CHK = SHK; the client and server must already share a key

What do you think of the idea of a session key?
Authentication

- In order to authenticate, the two ends have to prove their identity to each other
  - They have to show that they know the key
  - If they do that by exchanging the key – the key is compromised
- Kerberos is an example of such an authentication protocol
- Protects against replay attacks, offline password cracking, many others
  - The Moron’s guide to Kerberos: http://www.isi.edu/gost/brian/security/kerberos.html
Trusted Party Authentication – Kerberos

- Assumes that the trusted server shares a key with both A and B
- T is a token; protects against replay attacks
- L is the duration of the session key validity
Public Key Authentication

- If you encrypt a packet with B’s public key, only B can decrypt it.
- Only B can respond with the correct value.
- Mallory (man in the middle)?
Digital Signature

- Digital signature can be used for
  - Message Integrity
  - Non-repudiation: associate the message with the sender
    * This is an ongoing ethical/law battle – are digital signatures as good as signed documents to prove intent?
    * I will place two short articles on this issue on class webpage

- How can we do it?
  - A combination of message integrity and authentication?
Digital Signature – Approaches

- Digital Signature using RSA
  - Encrypt the message with your private key; only your public key will be able to decrypt it, verifying it is from you
  - Slow

- Keyed MD5
  - Sender: m + MD5(m+k) + E(k, private)
  - Receiver: recover k using public key; check MD5 signature

- MD5 with RSA signature
  - Sender: m + E(MD5(m), private)
  - Receiver decrypts signature with public key and matches MD5 signature
Public Key Distribution

- How do you make sure that the public key is correct (and not supplied by Mallory)
  - If I send it directly to you – Mallory can intercept
  - If we use a secure database for it – Mallory can intercept traffic to the database
  - IETF meetings: people used to gather public keys in person!
- Idea: use a trusted authority to sign the key – Certificates

- Certificates
  - Digitally signed document: “I certify that this public key belongs to this entity, signed X”
  - Similar to notary publics?
- Problems?
Tthing the Authorities

• How do you know the CA’s public key?
  – its certificate came with the browser
  – it’s certified by a CA whose certificate came with the browser

• ”I trust this certificate because I trust TC TrustCenter of Bucharest, Romania”
  – I dont think so

• ”I trust TC TrustCenter because both Microsoft and Netscape made a deal with it to include its certificate in their browsers”
  – Hmm, maybe ..
Certification Authority

- Certification Authority
  - Administrative entity that issues certificates
  - Useful only to someone who knows the CA’s public key
- Create a chain of trust
  - If X certifies a public K to belong to Y, then Y certifies that another belongs to Z, then there is a chain of trust from X to Z
  - To verify Z’s public key you need to know X’s and follow the chain
- Certificate revocation list
IPSec

- Protocol that works with both IPv4 and IPv6
  - RFCs 2401–2410

- Two pieces
  - Pair of protocols
    * authentication header providing access control, integrity, authentication and replay protection
    * Encapsulating security payload supporting above and adding confidentiality
  - Key management
    * Internet Security Association and Key Management Protocol