CprE 450/550x
Distributed Systems and Middleware

Security

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
April 29, 2004

Readings for Today's Lecture

- References
  - Chapter 8 of "Distributed Systems: Principles and Paradigms"
  - Ross Anderson, "Security Engineering"

3

Security Threats

- Leakage: An unauthorized party gains access to a service or data (eavesdropping).
- Tampering: Unauthorized change of data, tampering with a service
- Vandalism: Interference with proper operation, without gain to the attacker

4

Security Threats in Comm. Channels

- Eavesdropping – Obtaining copies of messages without authority.
- Masquerading – Sending or receiving messages with the identity of another principal.
- Message tampering – Intercepting messages and altering their contents before passing them onto the intended recipient.
- Replaying – Intercepting messages and sending them at a later time.
- Denial of Service Attack – flooding a channel or other resources with messages.

Security Policies & Mechanisms

- Security Policy indicates which actions each entity (user, data, service) is allowed or prohibited to take.
- Security Mechanism enforces the policy
  - Encryption: transform data to a form only understandable by authorized users.
  - Authentication: verify the claimed identity of a user, client, service, program, etc.
  - Authorization: verify access rights for an authenticated entity.
  - Auditing: make record of and check access to data and resources. Mainly an analysis tool to measure the success of security policies and mechanisms

Familiar Names for Principals in Security Protocols

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

7

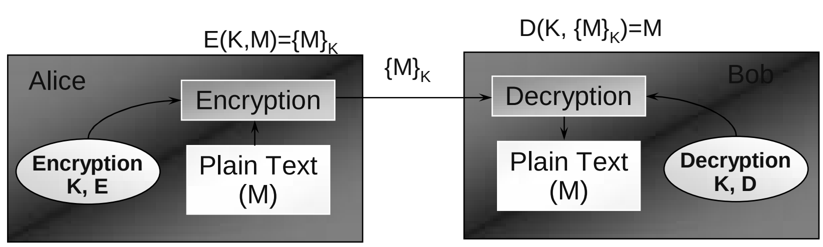
Cryptography Notations

K_A	Alice's secret key
K_B	Bob's secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice's private key (known only to Alice)
K_{Apub}	Alice's public key (published by Alice for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

8

Cryptography

- Encoding (encryption) of a message that can only be read (decryption) by a key.
- In shared key cryptography (symmetric cryptography) the sender and the recipient know the key, but no one else does.
  - **How do Alice and Bob get the shared key K_{AB} to begin with?**
- In public/private key pairs messages are encrypted with a published public key, and can only be decrypted by a secret private decryption key.

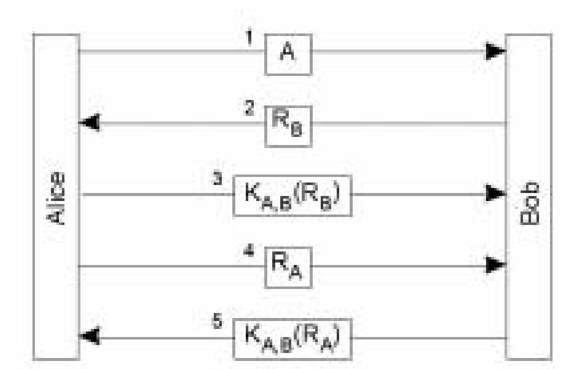


Authentication

- Use of cryptography for safeguarding communication between two principals.
- In direct authentication, the server uses a shared secret key to authenticate the client.
- In indirect authentication, a trusted authentication server provides a ticket to an authenticated client.
  - **The authentication server knows keys of principals and generates temporary shared keys.**
  - **In electronic commerce or wide area applications, public/private key pairs are used rather than shared keys.**

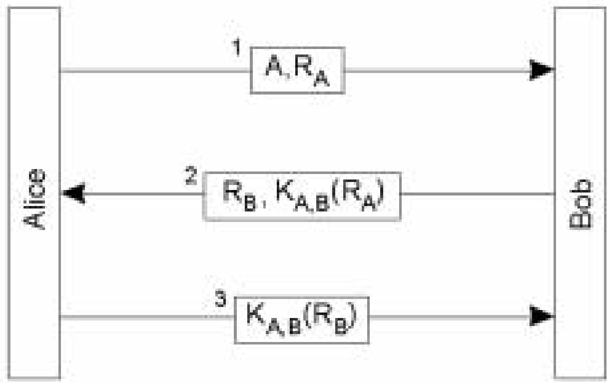
Direct Authentication

- Authentication based on a shared secret key.

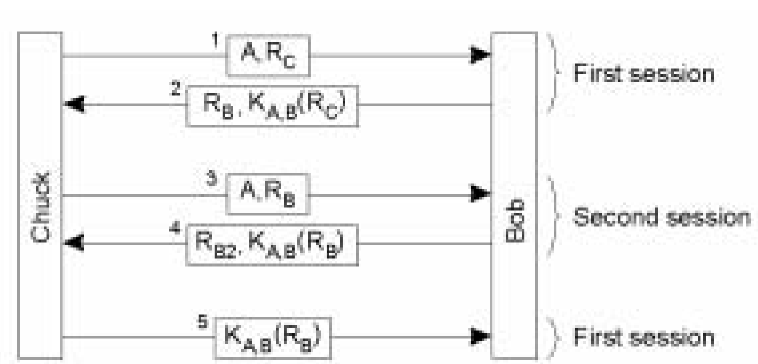


"Optimized" Direct Authentication

- Authentication based on a shared secret key, but using three instead of five messages.

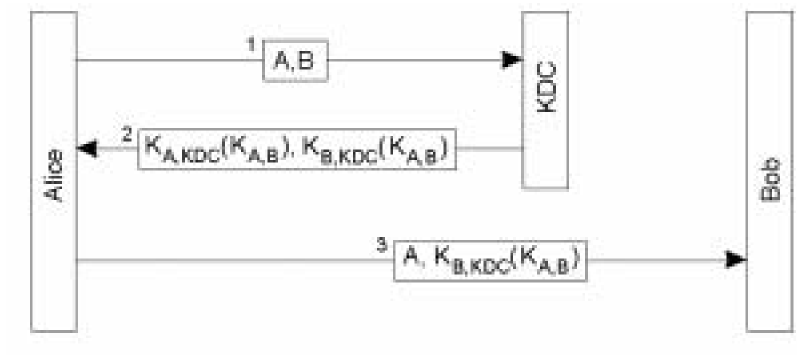


Reflection Attack

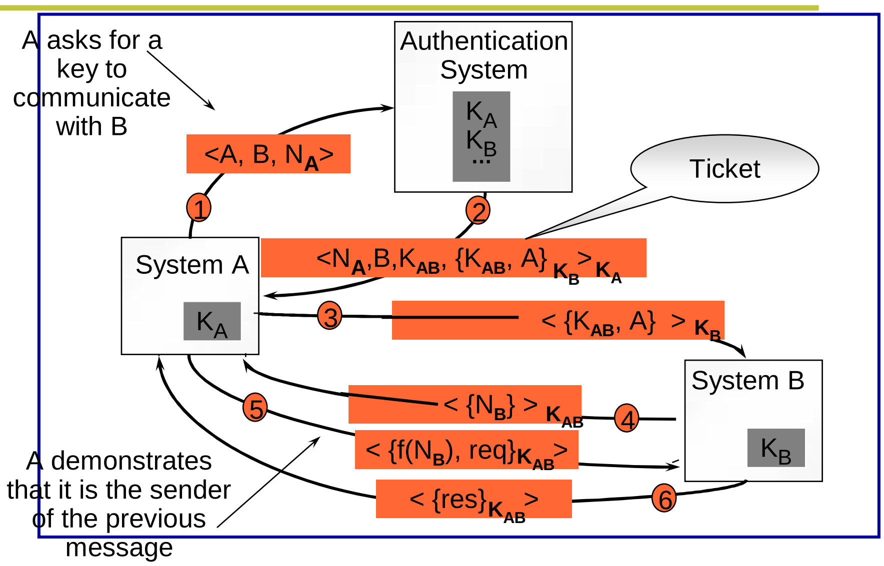


Authentication Using a Key Distribution Center

- Using a ticket and letting Alice set up a connection to Bob.



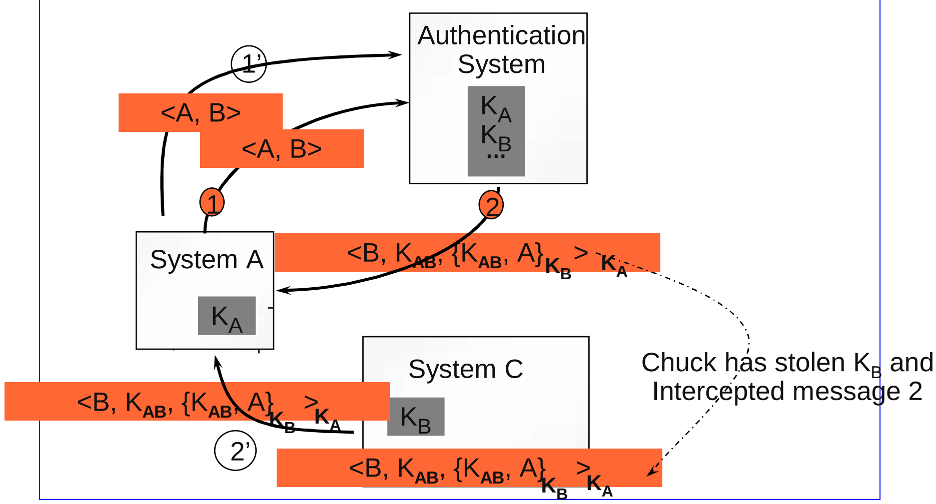
Needham-Schroeder Authentication



15

Why Do We Need Nonce N_A in Message 1?

Because we need to relate message 2 to message 1



16

Needham–Schroeder Secret-key Authentication Protocol

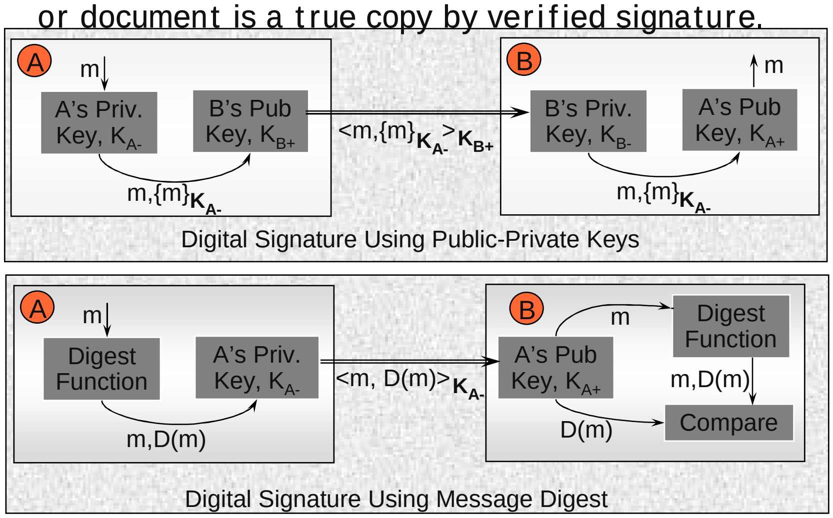
Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S->A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key.
3. A->B:	$\{K_{AB}, A\}_{K_B}$	A sends the 'ticket' to B.
4. B->A:	$\{N_B\}_{K_{AB}}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}_{K_{AB}}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

Kerberos Authentication

Read section 8.5 from text

Digital Signatures

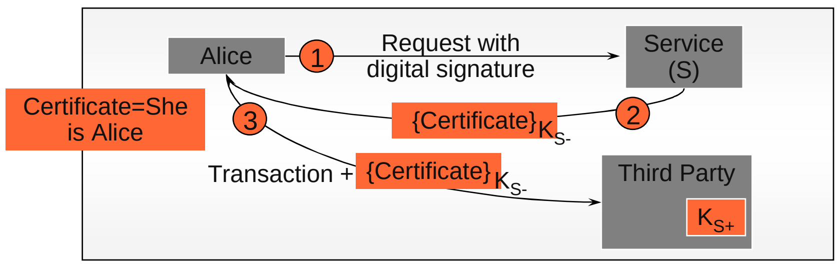
- Cryptography is also used to verify that a message or document is a true copy by verified signature.



19

Digital Certificates

- A digital certificate is a statement signed by a third party principal.
- To be useful, certificates must have:
  - **A standard format, for construction and interpretation**
  - **A protocol for constructing chains of certificates**
  - **A trusted authority at the end of the chain**



20

Alice's Bank Account Certificate

1. *Certificate type*	Account number
2. *Name*	Alice
3. *Account*	6262626
4. *Certifying authority*	Bob's Bank
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Alice may pretend to be the bank and create a new key pair, K_{B+}, K_{B}-

21

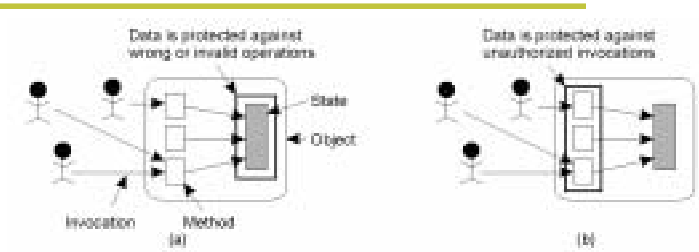
Public-Key Certificate for Bob's Bank

1. *Certificate type*	Public key
2. *Name*	Bob's Bank
3. *Public key*	K_{Bpub}
4. *Certifying authority*	Fred – The Bankers Federation
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

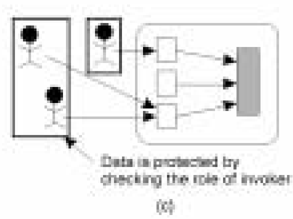
Eventually K_F-, K_F+ have to be obtained reliably.

22

Focus of Access Control



- Three approaches for protection against security threats
  - a) Protection against invalid operations
  - b) Protection against unauthorized invocations
  - c) Protection against unauthorized users



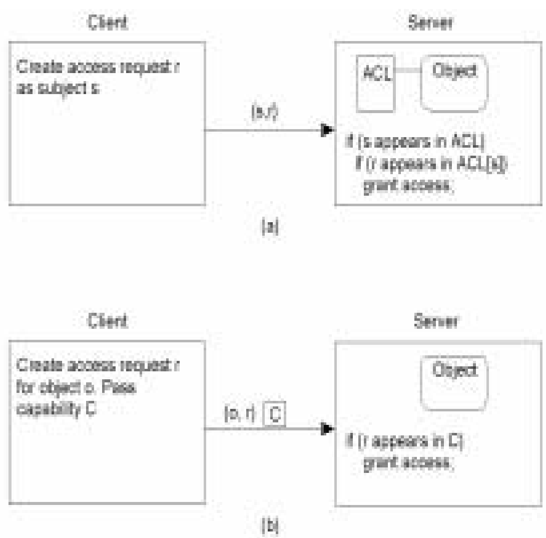
Access Control

- Control of access to resources of a server.
- A basic form of access control checks <principal, op, resource> requests for:
  - Authenticity of the principal or its credentials.
  - Access rights for the requested resource & op.
- Access control matrix M.
  - Each principal is represented by a row, and each resource object is represented by a column.
  - M[s,o] lists precisely what operations principal s can request to be carried out on resource o.
  - May be sparse.
- Access control list (ACL)
  - Each object maintains a list of access rights of principals, I.e., an ACL is some column in M with the empty entries left out.

Access Control Matrix

Comparison between ACLs and capabilities for protecting objects.

a) Using an ACL
b) Using capabilities.



Access Control

- The server may issue to each principal a list of capabilities.
  - **A list of capabilities corresponds to an entry in the access control matrix.**
- To reduce ACLs, the notion of protection domain is introduced.
  - A protection domain is a set of (object, access rights) pairs kept by a server.
  - Whenever a principal requests an operation to be carried out on an object, the access control monitor checks if the principal belongs to that domain, and then if the request is allowed for that object.
- Each principal can carry a certificate listing the groups it belongs to.
  - The certificate should be protected by a digital signature.

Firewalls

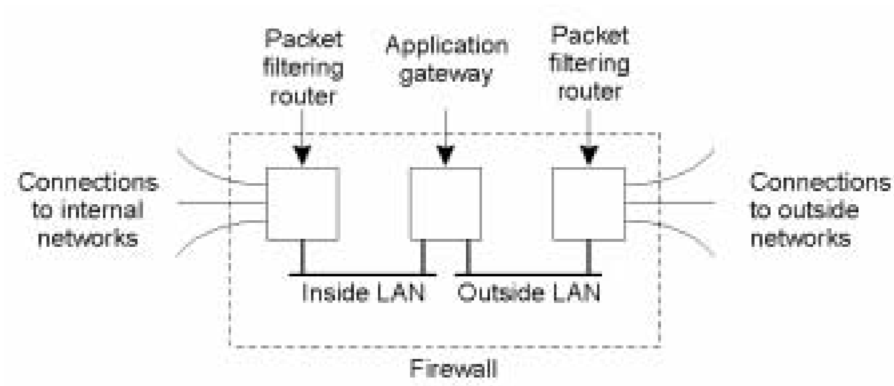
Firewall filtering can be done at diff. levels

- IP packet filtering: operates as a router and makes decisions as to whether or not to pass a packet based on its source/destination addresses.
  - The gateway on the outside LAN protects against incoming packets. The gateway on the inside LAN protects against outgoing packets.
- TCP gateway: checks all TCP connection requests and segment transmissions. TCP segments will be checked for correctness and may be routed to an application-level gateway for content checking.
- Application-level filtering (proxy gateway): inspects the content of incoming/outgoing messages.
  - To prevent applets to be downloaded to the inside LAN, all Web traffic could be directed through a Web proxy gateway. The gateway accepts regular HTTP requests, but may discard certain requests/pages.

27

Firewall Configuration

- A common implementation of a firewall.

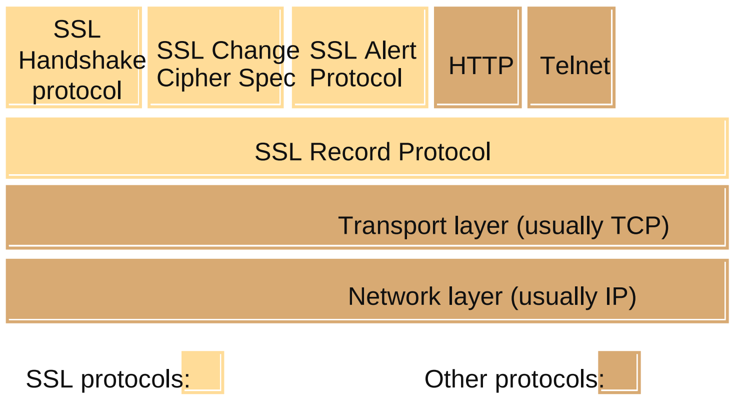


28

Secure Socket Layer Protocol

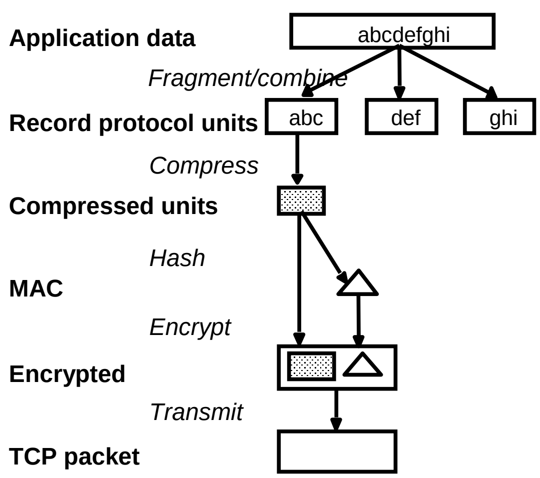
- SSL was developed by Netscape for electronic transaction security.
- A protocol layer is added below the application layer for:
  - **Negotiating encryption and authentication methods.**
  - **Bootstrapping secure communication**
- It consists of two layers:
  - **The Record Protocol Layer implements a secure channel by encrypting and authenticating messages**
  - **The Handshake Layer establishes and maintains a secure session between two nodes.**

SSL Protocol Stack



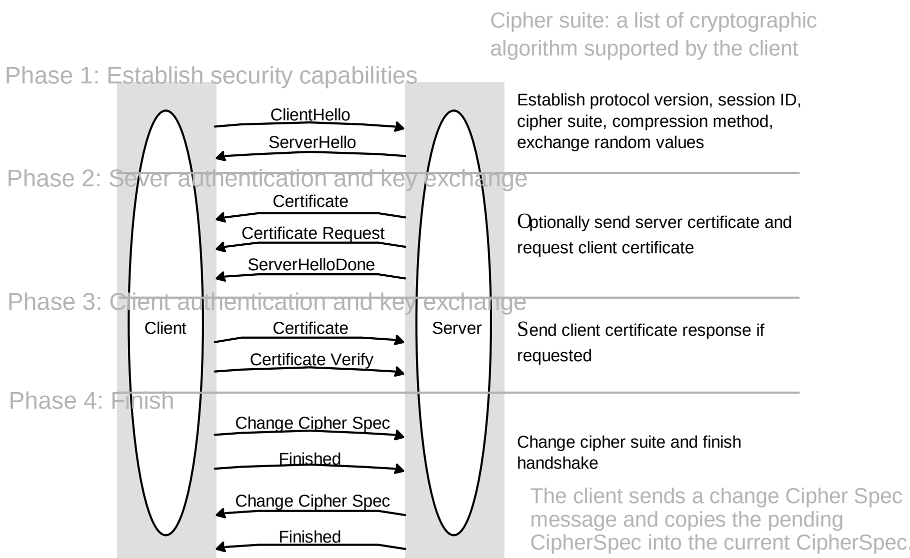
SSL Record Protocol

- The record protocol takes an application message to be transmitted,
  - fragments the data into manageable blocks,
  - optionally compresses the data,
  - computes a message authentication code (MAC),
  - encrypts and
  - adds a header.



31

SSL Handshake Protocol



CprE 450/550x
Distributed Systems and Middleware

Distributed File Systems and P2P Systems

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

April 29, 2004

33

Readings for Today's Lecture

- References
  - Chapter 10 of "Distributed Systems: Principles and Paradigms"
  - Paper list on Peer-to-Peer systems on the course page.

34

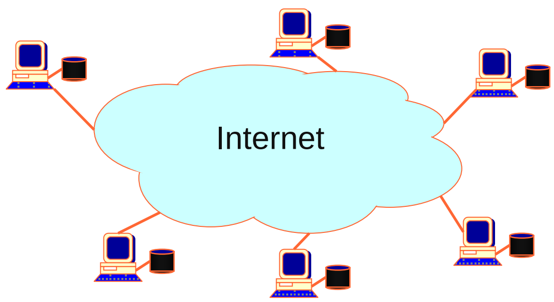
Distributed File Systems

Issue	NFS	Coda	Plan 9	xFS	SFS
Design goals	Access transparency	High availability	Uniformity	Serverless system	Scalable security
Access model	Remote	Up/Download	Remote	Log-based	Remote
Communication	RPC	RPC	Special	Active msgs	RPC
Client process	Thin/Fat	Fat	Thin	Fat	Medium
Server groups	No	Yes	No	Yes	No
Mount granularity	Directory	File system	File system	File system	Directory
Name space	Per client	Global	Per process	Global	Global
File ID scope	File server	Global	Server	Global	File system
Sharing sem.	Session	Transactional	UNIX	UNIX	N/S
Cache consist.	write-back	write-back	write-through	write-back	write-back
Replication	Minimal	ROWA	None	Striping	None
Fault tolerance	Reliable comm.	Replication and caching	Reliable comm.	Striping	Reliable comm.
Recovery	Client-based	Reintegration	N/S	Checkpoint & write logs	N/S
Secure channels	Existing mechanisms	Needham-Schroeder	Needham-Schroeder	No pathnames	Self-cert.
Access control	Many operations	Directory operations	UNIX based	UNIX based	NFS BASED

- A comparison between NFS, Coda, Plan 9, xFS. N/S indicates that nothing has been specified.

How Did it Start?

- A killer application: Naptser
  - Free music over the Internet
- Key idea: share the content, storage *and* bandwidth of individual (home) users

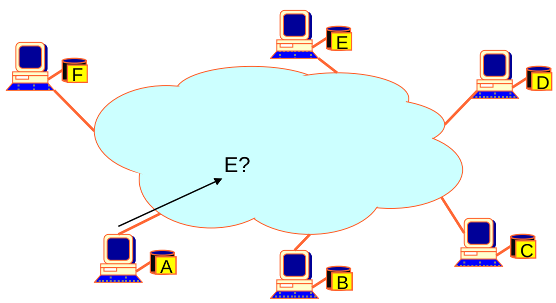


Model

- Each user stores a subset of files
- Each user has access (can download) files from all users in the system

Main Challenge

- Find where a particular file is stored



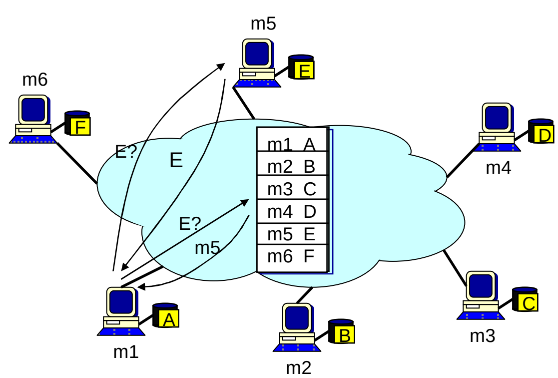
Other Challenges

- Scale: up to hundred of thousands or millions of machines
- Dynamicity: machines can come and go any time

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**
- Advantages:
  - **Simplicity, easy to implement sophisticated search engines on top of the index system**
- Disadvantages:
  - **Robustness, scalability (?)**

Napster: Example



41

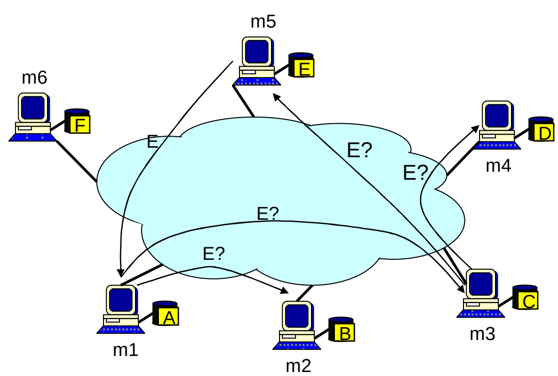
Gnutella

- Distribute file location
- Idea: flood the request
- How to find a file:
  - **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**
- Advantages:
  - **Totally decentralized, highly robust**
- Disadvantages:
  - **Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)**

42

Gnutella: Example

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



43

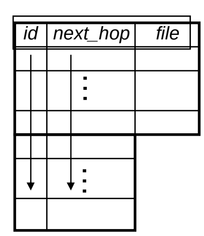
Freenet

- Addition goals to file location:
  - **Provide publisher anonymity, security**
  - **Resistant to attacks – a third party shouldn't be able to deny the access to a particular file (data item, object), even if it compromises a large fraction of machines**
- Architecture:
  - **Each file is identified by a unique identifier**
  - **Each machine stores a set of files, and maintains a "routing table" to route the individual requests**

44

Data Structure

- Each node maintains a common stack
  - *id* – file identifier
  - *next_hop* – another node that store the file id
  - *file* – file identified by *id* being stored on the local node
- Forwarding:
  - Each message contains the file *id* it is referring to
  - If file *id* stored locally, then stop;
  - If not, search for the "closest" *id* in the stack, and forward the message to the corresponding *next_hop*



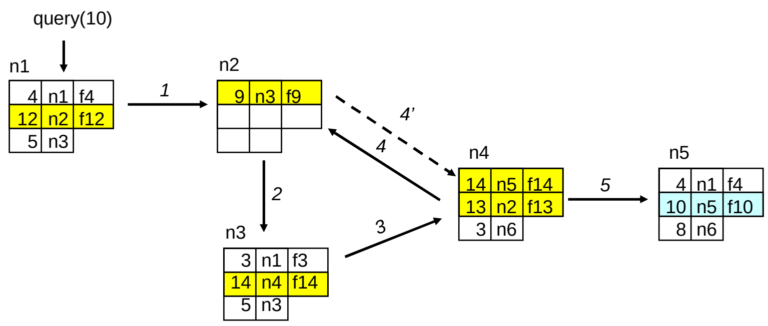
45

Query

- API: *file* = query(*id*);
- Upon receiving a query for document *id*
  - Check whether the queried file is stored locally
    - » If yes, return it
    - » If not, forward the query message
- Notes:
  - Each query is associated a TTL that is decremented each time the query message is forwarded; to obscure distance to originator:
    - » TTL can be initiated to a random value within some bounds
    - » When TTL=1, the query is forwarded with a finite probability
  - Each node maintains the state for all outstanding queries that have traversed it → help to avoid cycles
  - When file is returned, the file is cached along the reverse path

46

Query Example



- Note: doesn't show file caching on the reverse path

Insert

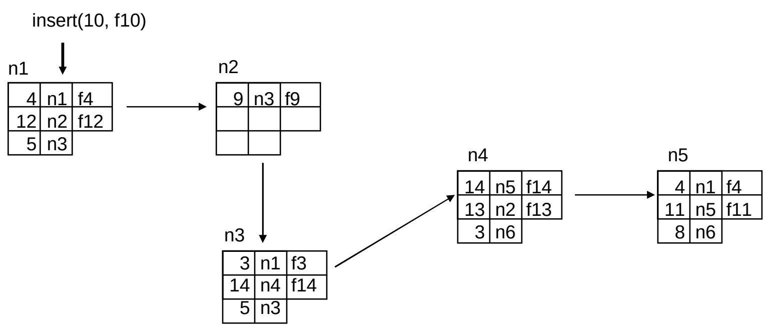
- API: insert(*id*, *file*);
- Two steps
  - Search for the file to be inserted
  - If not found, insert the file

Insert

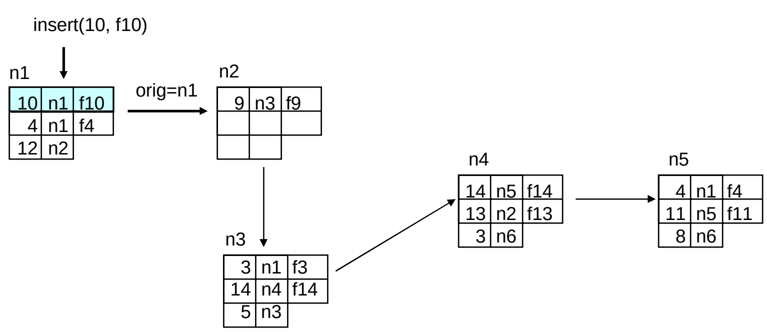
- Searching: like query, but nodes maintain state after a collision is detected and the reply is sent back to the originator
- Insertion
  - Follow the forward path; insert the file at all nodes along the path
  - A node probabilistically replace the originator with itself; obscure the true originator

Insert Example

- Assume query returned failure along "gray" path; insert f10



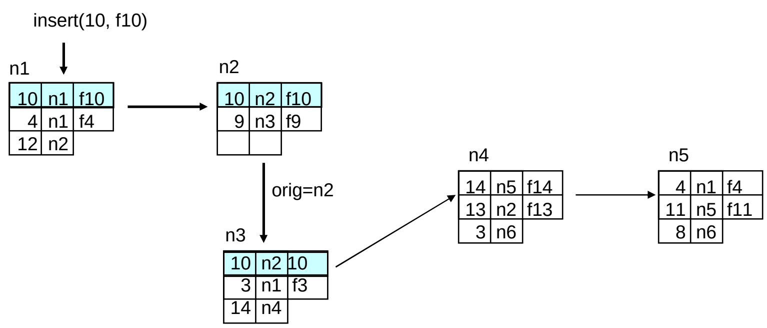
Insert Example



51

Insert Example

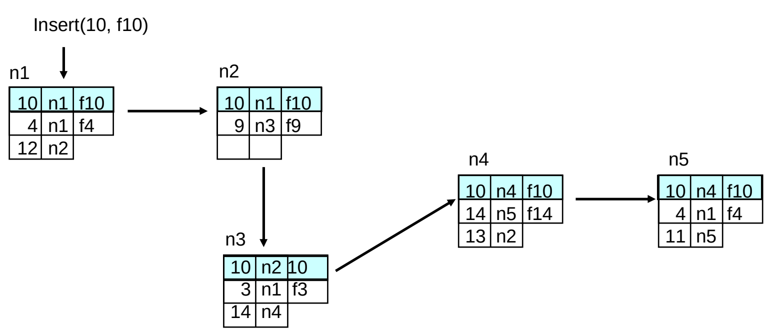
- n2 replaces the originator (n1) with itself



52

Insert Example

- n2 replaces the originator (n1) with itself



53

Freenet Properties

- Newly queried/inserted files are stored on nodes storing similar ids
- New nodes can announce themselves by inserting files
- Attempts to supplant or discover existing files will just spread the files

54

Freenet Summary

- Advantages
  - Provides publisher anonymity
  - Totally decentralize architecture → robust and scalable
  - Resistant against malicious file deletion
- Disadvantages
  - Does not always guarantee that a file is found, even if the file is in the network

Other Solutions to the Location Problem

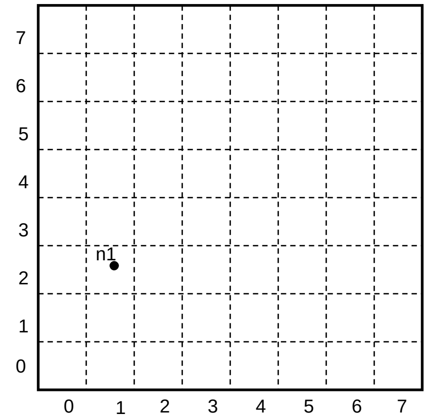
- Goal: make sure that an item (file) identified is always found
- Abstraction: a distributed hash-table data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Proposals
  - CAN, Chord, Kademlia, Pastry, Viceroy, Tapestry, etc

Content Addressable Network (CAN)

- Associate to each node and item a unique *id* in an *d*-dimensional Cartesian space
- Goals
  - **Scales to hundreds of thousands of nodes**
  - **Handles rapid arrival and failure of nodes**
- Properties
  - **Routing table size O(*d*)**
  - **Guarantees that a file is found in at most $d*n^{1/d}$ steps, where *n* is the total number of nodes**

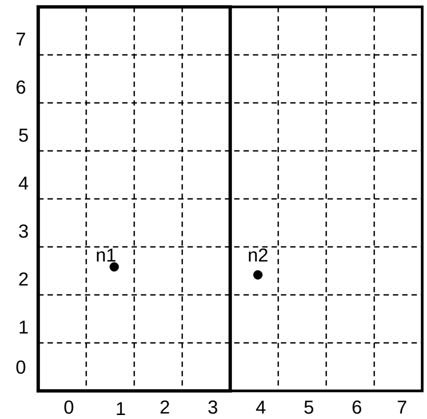
CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
  - Node n1:(1, 2) first node that joins → cover the entire space



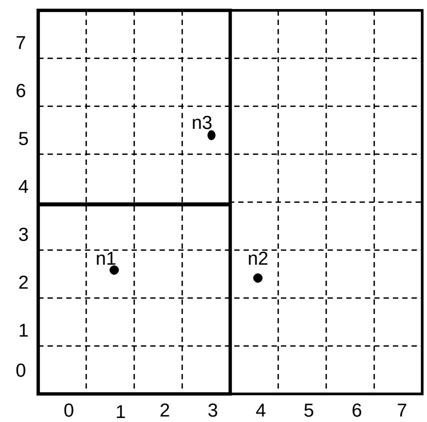
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



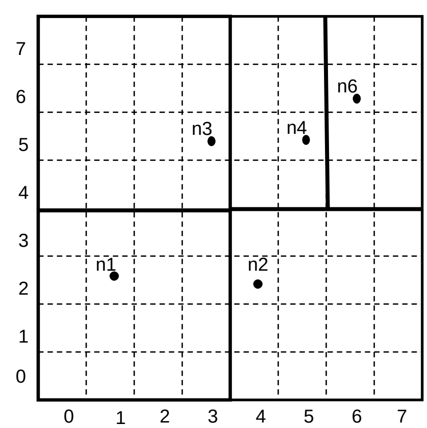
CAN Example: Two Dimensional Space

- Node n3:(3, 5) joins → space is divided between n1 and n2



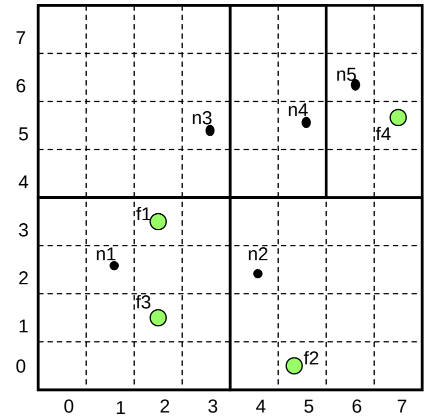
CAN Example: Two Dimensional Space

- Nodes n4:(5, 5) and n5:(6,6) join



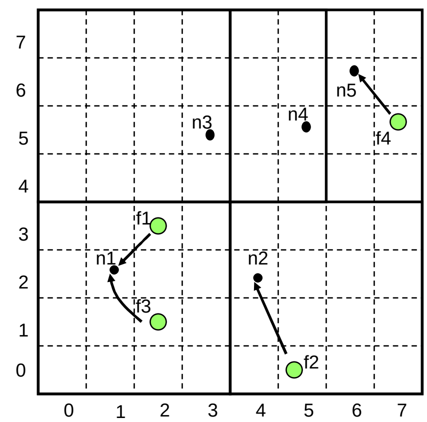
CAN Example: Two Dimensional Space

- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5);n5:(6,6)
- Items: f1:(2,3); f2:(5,1); f3:(2,1); f4:(7,5);



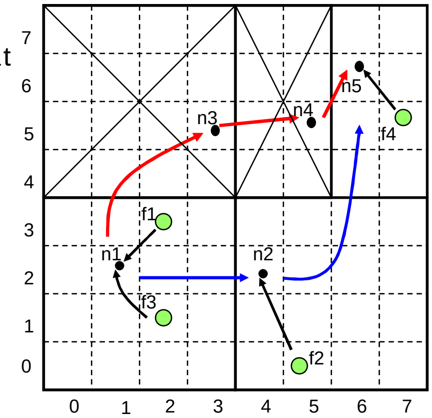
CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space



CAN: Query Example

- Each node knows its neighbors in the *d*-space
- Forward query to the neighbor that is closest to the query *id*
- Example: assume n1 queries f4
- Can route around some failures



Node Failure Recovery

- Simple failures
  - Know your neighbor's neighbors
  - When a node fails, one of its neighbors takes over its zone
- More complex failure modes
  - Simultaneous failure of multiple adjacent nodes
  - Scoped flooding to discover neighbors
  - Hopefully, a rare event

Chord

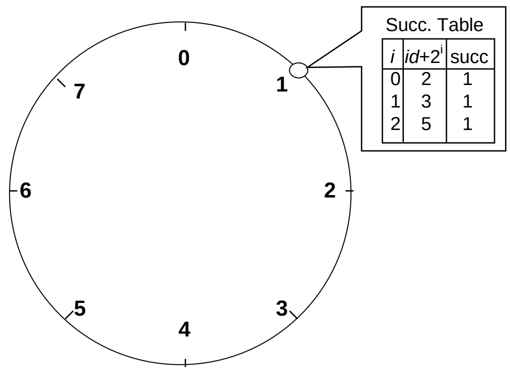
- Associate to each node and item a unique *id* in an *uni-* dimensional space
- Goals
  - Scales to hundreds of thousands of nodes
  - Handles rapid arrival and failure of nodes
- Properties
  - Routing table size $O(\log(N))$, where N is the total number of nodes
  - Guarantees that a file is found in $O(\log(N))$ steps

Data Structure

- Assume identifier space is $0..2^m$
- Each node maintains
  - Finger table
    - Entry i in the finger table of n is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the succesor node of *id*

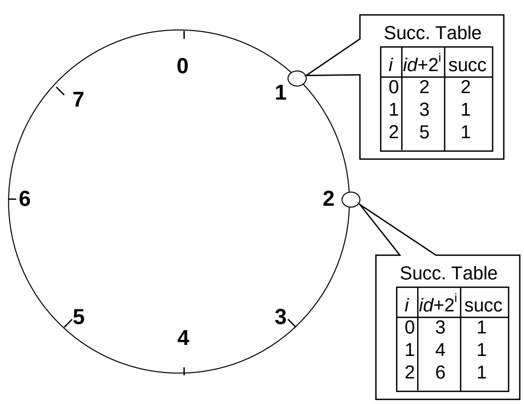
Chord Example

- Assume an identifier space 0..8
- Node n1:(1) joins→all entries in its finger table are initialized to itself



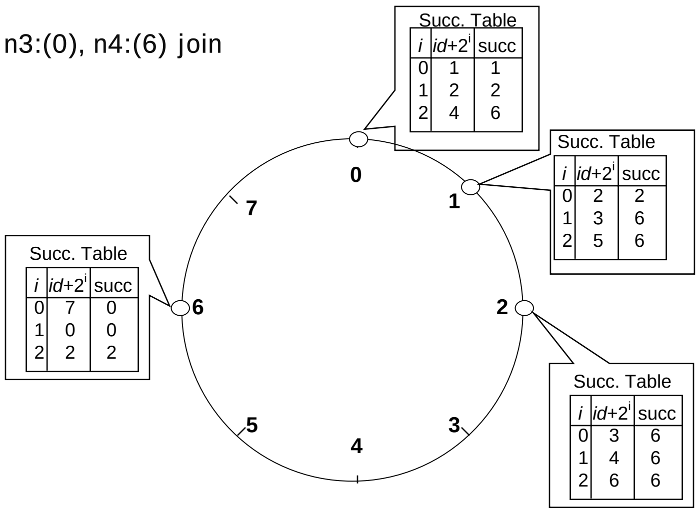
Chord Example

- Node n2:(3) joins



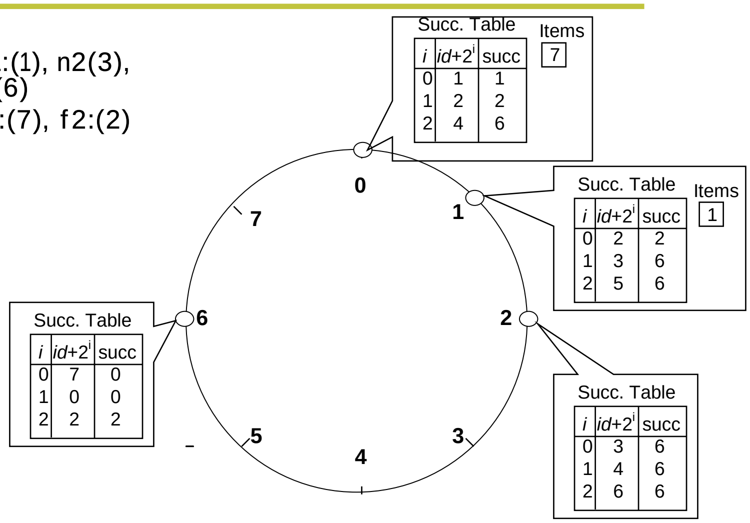
Chord Example

- Nodes n3:(0), n4:(6) join



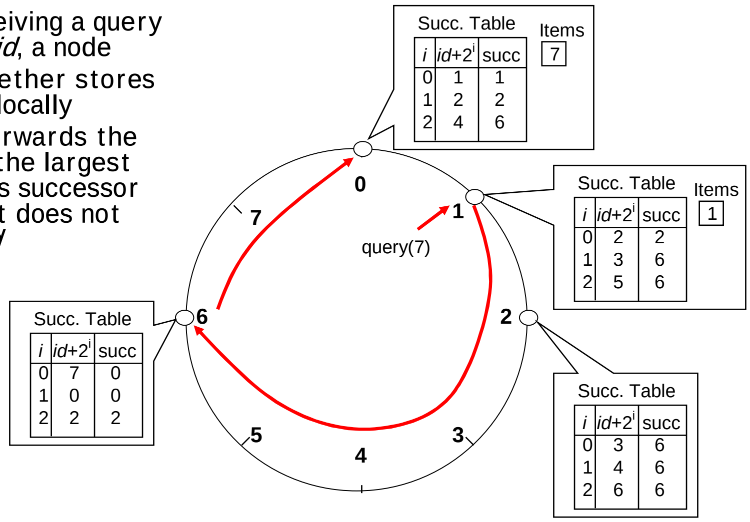
Chord Examples

- Nodes: n1:(1), n2(3), n3(0), n4(6)
- Items: f1:(7), f2:(2)



Query

- Upon receiving a query for item *id*, a node
- Check whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



Node Joining

- Node n joins the system:
  - **n picks a random identifier, id**
  - **n performs n' = lookup(id)**
  - **n->successor = n'**

State Maintenance: Stabilization Protocol

- Periodically node n
  - Asks its successor, n', about its predecessor n"
  - If n" is between n' and n"
    - n->successor = n"
    - notify n" that n its predecessor
- When node n" receives notification message from n
  - If n is between n"->predecessor and n", then
    - n"->predecessor = n
- Improve robustness
  - Each node maintain a successor list (usually of size 2*log N)

CAN/Chord Optimizations

- Weight neighbor nodes by RTT
  - When routing, choose neighbor who is closer to destination with lowest RTT from me
  - Reduces path latency
- Multiple physical nodes per virtual node
  - Reduces path length (fewer virtual nodes)
  - Reduces path latency (can choose physical node from virtual node with lowest RTT)
  - Improved fault tolerance (only one node per zone needs to survive to allow routing through the zone)
- Several others

Conclusions

- Distributed Hash Tables are a key component of scalable and robust overlay networks
- CAN: O(d) state, O(d*n1/d) distance
- Chord: O(log n) state, O(log n) distance
- Both can achieve stretch < 2
- Simplicity is key
- Services built on top of distributed hash tables
  - p2p file storage, i3 (chord)
  - multicast (CAN, Tapestry)
  - persistent storage (OceanStore using Tapestry)

Thanks to all of you!!!

Wish you have a happy and safe Summer!

See you next Monday!