CS 552
Peer 2 Peer Networking

R. Martin
Credit slides from B. Richardson, I. Stoica, M. Cuenca
Peer to Peer

• Outline
• Overview
• Systems:
  – Gnutella
  – Freenet
  – Chord
  – PlanetP
Why Study P2P

• Huge fraction of traffic on networks today
  – >=50%

• Exciting new applications

• Next level of resource sharing
  – Vs. timesharing, client-server, P2P
  – E.g. Access 10’s-100’s of TB at low cost.
P2P usage

- CMU network (external to world), 2003
- 47% of all traffic was easily classifiable as P2P
- 18% of traffic was HTTP
- Other traffic: 35%
  - Believe ~28% is port-hopping P2P
- Other sites have a similar distribution
Big Picture

- **Gnutella**
  - Focus is simple sharing
  - Using simple flooding
- **Bit torrent**
  - Designed for high bandwidth
- **PlanetP**
  - Focus on search and retrieval
  - Creates global index on each node via controlled, randomized flooding
- **Cord**
  - Focus on building a distributed hash table (DHT)
  - Finger tables
Other P2P systems

• Freenet:
  – Focus privacy and anonymity
  – Builds internal routing tables
• KaaZa
• eDonkey
• Napster
  – Success started the whole craze
Key issues for P2P systems

• **Join/leave**
  – How do nodes join/leave? Who is allowed?

• **Search and retrieval**
  – How to find content?
  – How are metadata indexes built, stored, distributed?

• **Content Distribution**
  – Where is content stored? How is it downloaded and retrieved?
Search and Retrieval

• Basic strategies:
  – Flooding the query
  – Flooding the index
  – Routing the query

• Different tradeoffs depending on application
  – Robustness, scalability, legal issues
Flooding the Query (Gnutella)

Pros: highly robust. Cons: Huge network traffic

Key=title
Value=mp3
Pros: Robust. Cons: Index size
Routing the Query (Chord)
What is Gnutella?

- Gnutella is a protocol for distributed search
- Each node in a Gnutella network acts as both a client and server
- Peer to Peer, decentralized model for file sharing
- Any type of file can be shared
- Nodes are called “Servents”
What do Servents do?

- Servents “know” about other Servents
- Act as interfaces through which users can issue queries and view search results
- Communicate with other Servents by sending “descriptors”
Descriptors

- Each descriptor consists of a header and a body.

- The header includes (among other things)
  - A descriptor ID number
  - A Time-To-Live number

- The body includes:
  - Port information
  - IP addresses
  - Query information
  - Etc… depending on the descriptor
Gnutella Descriptors

- **Ping**: Used to discover hosts on the network.
- **Pong**: Response to a Ping
- **Query**: Search the network for data
- **QueryHit**: Response to a Query. Provides information used to download the file
- **Push**: Special descriptor used for sharing with a firewalled servant
Routing

• Node forwards Ping and Query descriptors to all nodes connected to it

• Except:
  – If descriptor’s TTL is decremented to 0
  – Descriptor has already been received before

• Loop detection is done by storing Descriptor ID’s

• Pong and QueryHit descriptors retrace the exact path of their respective Ping and Query descriptors
Note: Ping works essentially the same way, except that a Pong is sent as the response.
Joining a Gnutella Network

• Servent connects to the network using TCP/IP connection to another servent.

• Could connect to a friend or acquaintance, or from a “Host-Cache”.

• Send a Ping descriptor to the network

• Hopefully, a number of Pongs are received
Querying

- Servent sends **Query** descriptor to nodes it is connected to.

- Queried Servents check to see if they have the file.
  - If query match is found, a **QueryHit** is sent back to querying node
Downloading a File

• File data is never transferred over the Gnutella network.

• Data transferred by direct connection

• Once a servent receives a QueryHit descriptor, it may initiate the direct download of one of the files described by the descriptor’s Result Set.

• The file download protocol is HTTP. Example:

```plaintext
GET /get/<File Index>/<File Name>/ HTTP/1.0
Connection: Keep-Alive
Range: bytes=0-
User-Agent: Gnutella
```
Direct File Download

A
Query

B
TCP/IP Connection
Query
QueryHit

C
Overall:

- Simple Protocol
- Not a lot of overhead for routing
- Robustness?
  - No central point of failure
  - However: A file is only available as long as the file-provider is online.
- Vulnerable to denial-of-service attacks
Overall

• Scales poorly: Querying and Pinging generate a lot of unnecessary traffic

• Example:
  – If TTL = 10 and each site contacts six other sites
  – Up to $10^6$ (approximately 1 million) messages could be generated.

  – On a slow day, a GnutellaNet would have to move 2.4 gigabytes per second in order to support numbers of users comparable to Napster. On a heavy day, 8 gigabytes per second (Ritter article)

• Heavy messaging can result in poor performance
PlanetP Introduction

• 1st generation of P2P applications based on ad-hoc solutions
  – File sharing (Kazaa, Gnutella, etc), Spare cycles usage (SETI@Home)

• More recently, many projects are focusing on building infrastructure for large scale key-based object location (DHTS)
  – Chord, Tapestry and others
  – Used to build global file systems (Farsite, Oceanstore)

• What about content-based location?
Goals & Challenges

• Provide content addressing and ranking in P2P
  – Similar to Google/ search engines
  – Ranking critical to navigate terabytes of data

• Challenges
  – Resources are divided among large set of heterogeneous peers
  – No central management and administration
  – Uncontrolled peer behavior
  – Gathering accurate global information is too expensive
The PlanetP Infrastructure

• Compact global index of shared information
  – Supports resource discovery and location
  – Extremely compact to minimize global storage requirement
  – Kept loosely synchronized and globally replicated

• Epidemic based communication layer
  – Provides efficient and reliable communication despite unpredictable peer behaviors
  – Supports peer discovery (membership), group communication, and update propagation

• Distributed information ranking algorithm
  – Locate highly relevant information in large shared document collections
  – Based on TFxIDF, a state-of-the-art ranking technique
  – Adapted to work with only partial information
Using PlanetP

• **Services provided by PlanetP:**
  – **Content addressing and ranking**
    • Resource discovery for adaptive applications
  – **Group membership management**
    • Close collaboration
  – **Publish/Subscribe information propagation**
    • Decoupled communication and timely propagation
  – **Group communication**
    • Simplify development of distributed apps.
Global Information Index

- Each node maintains an index of its content
  - Summarize the set of terms in its index using a Bloom filter
- The global index is the set of all summaries
  - Term to peer mappings
  - List of online peers
  - Summaries are propagated and kept synchronized using gossiping
Epidemic Comm. in P2P

• Nodes push and pull randomly from each others
  – Unstructured communication ⇒ resilient to failures
  – Predictable convergence time

• Novel combination of previously known techniques
  – Rumoring, anti-entropy, and partial anti-entropy
    • Introduce partial anti-entropy to reduce variance in propagation
ten time for dynamic communities
  – Batch updates into communication rounds for efficiency
  – Dynamic slow-down in absence of updates to save bandwidth
Content Search in PlanetP

Local lookup

Rank nodes

Contact candidates

Rank results

Query

Global Directory

<table>
<thead>
<tr>
<th>Nickname</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Bob</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Charles</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Diane</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Edward</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Fred</td>
<td>$[K_1,..,K_n]$</td>
</tr>
<tr>
<td>Gary</td>
<td>$[K_1,..,K_n]$</td>
</tr>
</tbody>
</table>

Bob

Diane

Fred

Diane

Fred

Bob

File_3
File_1
File_2

STOP
Results Ranking

• The Vector Space model
  – Documents and queries are represented as k-dimensional vectors
    • Each dimension represents the relevance or weight of the word for the document
  – The angle between a query and a document indicates its similarity
  – Does not requires links between documents

• Weight assignment (TFxIDF)
  – Use Term Frequency (TF) to weight terms for documents
  – Use Inverse Document Frequency (IDF) to weight terms for query
  – Intuition
    • TF indicates how relevant a document is to a particular concept
    • IDF gives more weight to terms that are good discriminators between documents
Using TFxIDF in P2P

- Unfortunately IDF is not suited for P2P
  - Requires term to document mappings
  - Requires a frequency count for every term in the shared collection
- Instead, use a two-phase approximation algorithm
- Replace IDF with IPF (Inverse Peer Frequency)
  - $IPF(t) = f(\text{No. Peers/Peers with documents containing term } t)$
  - Individuals can compute a consistent global ranking of peers and documents without knowing the global frequency count of terms
- Node ranking function
  
  $$Rank_i(Q) = \sum_{t \in Q \land t \in BF_i} IPF_t$$
Pruning Searches

• Centralized search engines have index for entire collection
  – Can rank entire set of documents for each query
• In a P2P community, we do not want to contact peers that have only marginally relevant documents
  – Use adaptive heuristic to limit forwarding of query in 2\textsuperscript{nd}-phase to only a subset of most highly ranked peers
Evaluation

• Answer the following questions
  – What is the efficacy of our distributed ranking algorithm?
  – What is the storage cost for the globally replicated index?
  – How well does gossiping work in P2P communities?

• Evaluation methodology
  – Use a running prototype to validate and collect micro benchmarks (tested with up to 200 nodes)
  – Use simulation to predict performance on big communities
  – We model peer behavior based on previous work and our own measurements from a local P2P community of 4000 users

• Will show sampling of results from paper
Ranking Evaluation I

• We use the AP89 collection from TREC
  – 84678 documents, 129603 words, 97 queries, 266MB
  – Each collection comes with a set of queries and relevance judgments

• We measure recall (R) and precision (P)

\[
R(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. relevant docs. in collection}}
\]

\[
P(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. docs. presented to the user}}
\]
Ranking Evaluation II

- Results intersection is 70% at low recall and gets to 100% as recall increases
- To get 10 documents, PlanetP contacted 20 peers out of 160 candidates
Global Index Space Efficiency

• TREC collection (pure text)
  – Simulate a community of 5000 nodes
    • Distribute documents uniformly
  – 944,651 documents taking up 3GB
  – 36MB of RAM are needed to store the global index
  – This is 1% of the total collection size

• MP3 collection (audio + tags)
  – Using previous result but based on Gnutella measurements
  – 3,000,000 MP3 files taking up 14TB
  – 36MB of RAM are needed to store the global index
  – This is 0.0002% of the total collection size
Data Propagation

Propagation speed experiment (DSL)  Arrival and departure experiment (LAN)
• Explored the design of infrastructural support for a rich set of P2P applications
  – Membership, content addressing and ranking
  – Scale well to thousands of peers
  – Extremely tolerant to unpredictable dynamic peer behaviors
• Gossiping with partial anti-entropy is reliable
  – Information always propagate everywhere
  – Propagation time has small variance
• Distributed approximation of TFxIDF
  – Within 11% of centralized implementation
  – Never collect all needed information in one place
  – Global index on average is only 1% of data collection
  – Synchronization of global index only requires 50 B/sec
BitTorrent: History

• In 2002, Bram Cohen debuted BitTorrent
• Key Motivation:
  – Popularity exhibits temporal locality, Flash Crowds)
  – E.g., Slashdot effect, CNN on 9/11, new release
• Focused on Efficient *Fetching*, not *Searching*:
  – Distribute the *same* file to all peers
  – Single publisher, multiple downloaders
• Has real publishers:
  – Blizzard Entertainment using it to distribute the beta of their new game
BitTorrent: Overview

• Swarming:
  – **Join**: contact centralized “tracker” server, get a list of peers.
  – **Publish**: Run a tracker server.
  – **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  – **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
BitTorrent Concepts

- Tracker
  - Lists of peers sharing the file
- .torrent files (.tor)
  - File length, name, hashing info, URL of tracker
- Seed
  - A peer with a copy of the file.
- Peers
  - Downloaders
  - Uploaders
BitTorrent Strategy

• Users finds content (.torrent of the content) via external search
  – E.g. google.
• Peer contacts tracker
• Peer gets random list of other peers
• Peer starts downloading & uploading
BitTorrent: Publish/Join

Diagram showing the network topology with a central tracker node connected to multiple peer nodes.
BitTorrent: Fetch
BitTorrent Design Decisions

- Distribution topology:
  - Random Graph vs.
    - Trees (lose bandwidth of leaves)
    - Hash codes

- Data encoding:
  - 2-tier piecewise w/ hash code checksum vs:
    - Erasure codes
    - Turbo codes
Piece & Sub-Piece Selection

- Pieces (256Kb) and sub-pieces (16Kb)
- Download single piece at once
  - Pipeline 5 sub-pieces
- Uses Rarest Piece First (RPF) selection strategy.
  - Random piece for the 1st one, avoid bottlenecks
- Endgame mode
  - Request all remaining sub-pieces too all peers when downloading last set
BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - “I’ll share with you if you share with me”
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
  - Similar to: Prisoner’s Dilemma
- Approximates Pareto Efficiency
  - Game Theory: “No change can make anyone better off without making others worse off”
BitTorrent Concepts

• Tracker
  – Lists of peers sharing the file
• .torrent files (.tor)
  – File length, name, hashing info, URL of tracker
• Seed
  – A peer with a copy of the file.
• Peers
  – Downloaders
  – Uploaders
BitTorrent Strategy

• Users finds content (.torrent of the content) via external search
  – E.g. google.
• Peer contacts tracker
• Peer gets random list of other peers
• Peer starts downloading & uploading
BitTorrent Design Descisions

- **Distribution topology:**
  - Random Graph vs.
    - Trees (lose bandwidth of leaves)
    - Hash codes

- **Data encoding:**
  - 2-tier piecewise w/ hash code checksum vs:
    - Erasure codes
    - Turbo codes
Piece & Sub-Piece Selection

- Pieces (256Kb) and sub-pieces (16Kb)
- Download single piece at once
  - Pipeline 5 sub-pieces
- Uses Rarest Piece First (RPF) selection strategy.
  - Random piece for the 1st one, avoid bottlenecks
- Endgame mode
  - Request all remaining sub-pieces too all peers when downloading last set
Cooperative Strategy

• Keep freeloaders from dominating
• Choking:
  – Maintain connection, but refuse to upload
• Keep top 4 downloaders unchoked
  – Compute rate over 20 second window
  – Recompute set every 10 seconds
• Optimistic unchoke
  – Every 3 periods (30 sec), rotate peer to unchoke
• Anti-snubbing
  – if >1 minute no piece, “snubbed”, -> choke
  – Can perform optimistic unchoke
BitTorrent: Summary

• Pros:
  – Works reasonably well in practice
  – Gives peers incentive to share resources; avoids freeloaders

• Cons:
  – Pareto Efficiency relative weak condition
  – Central tracker server needed to bootstrap swarm (is this really necessary?)
Chord

• Associate to each node and item a unique \emph{id} in an \emph{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 successor node of $id$
Hashing Keys to Nodes

Node 105

N105

Key 5

K5

N32

K20

Circular 7-bit ID space

K80

N90
Basic Lookup

N105 \rightarrow N120 \rightarrow N10 \rightarrow N32 \rightarrow N60 \rightarrow N90 \rightarrow K80

"N90 has K80"

"Where is key 80?"
Lookup Algorithm

Lookup(my-id, key-id)
    n = my successor
    if my-id < n < key-id
        Lookup(id) on node n // goto next hop
    else
        return my successor // found the correct node

• Correctness depends only on successors
• O(N) lookup time, but we can do better
Shortcutting to \( \log(N) \) time

Finger table
Shortcutting

Finger $i$ point to successor $n+2i$
Basic Chord algorithm

Lookup(my-id, key-id)
  look in local finger table for
  highest node n such that my-id < n < key-id
  if n exists
    Lookup(id) on node n // goto next hop
  else
    return my successor // found the correct node
Chord Example

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

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
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)
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 is 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)
Freenet Outline

- Introduction
- Freenet Basics
- Architecture
- File Storage
- Keys
- Requests
- Responses
- Adding a new peer
- Inserting Files
- Additional Readings
Introduction

• What is Freenet?
  – P2P system
  – Giant virtual hard drive
  – Provides a service for storing and retrieving files anonymously over the internet

• Goal of Freenet
  – Anonymity
    • For both file provider and requester
    • Number one design goal of Freenet
    • An absolute must for freedom of speech
      – Difficult to censor material if you do not know who is uploading it, who is requesting it, and where it is stored
Freenet

• Each user provides to the network
  – Bandwidth
    • For transmitting files
    • Routing requests for files
  – Hard drive space for storing files
    • Called a “data store” in Freenet
  – All Peers are equal
    • No nodes function as supernodes

• Uses for Freenet
  – Publishing web sites (Freesites)
    • E.g. “Banned” books
  – Message boards
  – Games
  – File sharing
File Storage

• Unlike other file sharing applications:
  - The user of a node has no control over or knowledge of what files their node stores
  - No user knows the identity of a node that provides a file they have requested or knows the identity of a node that has requested a file from them
    • Routing requests and responses through multiple nodes helps
    • Address of previous node is removed after each hop of a response
  - All files in each nodes data store are encrypted
    • No user of a node knows the contents of the files they are storing
    • This is done to protect the owner of each node from responsibility for the type of content stored in their data store
• Files remain in the system based on demand
  – Popular files will spread to many nodes
    • Each requested file located, will be copied to every node it
      passes through on the path from the source node to the
      requestor node
  – Rarely accessed files will slowly be removed from the
    network as room is required for new files
    • As a node runs out of space, files will be deleted in order of
      least recently requested to make room
    • Rarely requested files will ONLY be removed if space
      becomes limited
Keys

• In Freenet all files are requested based on a key assigned to the file when it was inserted into the network
  – Three types of keys:
    • SSK – signed subspace key
    • KSK – keyword signed key
    • CHK – content hash key

• Each Freenet key has the following structure
  – “freenet:” is the standard prefix
  – First three chars state key type: SSK, KSK, CHK
  – “@” symbol separates the key type from the rest of the message
  – Then a long set of characters used to identify the file
  – Example:
    • freenet:KSK@papers/p2p/freenet/keys
KSK – Keyword Signed Key

• Most basic type of file key
  – Easiest to use of all the key types
    • Descriptive set of words used to identify the file

• Steps to create the key:
  – 1. User writes a string describing the file
     – i.e., papers/p2p/freenet/keys.doc
  – 2. Specify that the key is of type KSK
  – 3. Add the prefix “freenet:”
  – 4. Specify the location of the file to insert
    • i.e., freenet:KSK@papers/freenet/keys.doc freenet_keys.doc
• **Advantages**
  – Only the file description needs to be published
  – Easy to pass on to others and remember

• **Disadvantages**
  – No namespace is used
    • No way to prevent two users from inserting two completely different files with the same description
  – Users can abuse the names of popular files by inserting their file with the same name
    • This is made possible because the file description is published
  – Dictionary attacks
SSK – Signed Subspace Key

• Problems with KSK:
  – Duplicate file names - no protection
  – KSK@papers/brian/freenet.doc, KSK@papers/brian/p2p.doc
    • No way for others to know if these two files were both uploaded by me!
• SSK - Allows for declaring of namespaces
  – Randomly generated public/private key pair
  – Used to identify the users own subspace
  – To get the key for the subspace:
    • 1. Public key is hashed
    • 2. String that describes the file is hashed
    • 3. (1) XOR (2)
    • 4. (3) is hashed
    • 5. (4) is encrypted using the file description
SSK – Signed Subspace Key

- **Private Key**
  - Only the person who possesses the private key can insert files to the namespace in the network
  - Allows others to ensure a file was posted by a certain person
  - Insert example
    - SSK@my_private_key/papers/brian/freenet.doc

- **Public Key**
  - Allows users to retrieve the file from the network
  - Request example
    - SSK@my_public_key/papers/brian/freenet.doc
  - Guarantees the requester that the file is from my subspace

- **Disadvantage - Updating of existing files**
CHK – Content Hash Key

• Key
  – Creates a unique key based on hashing the files content

• Steps to create the key
  – 1. Hash the content of the file to generate the key
  – 2. File is encrypted with a randomly generated key

• To allow others to retrieve the file need to give them
  – Content hash key
  – Decryption key
  – i.e., CHK@AN2lV5VzK9TdWHarfIYmv-xtf2ELAwI,ymQiGP7s4ZFR9FiAgV-ZpQ

• Can be used in combination with an SSK subspace
  – Two step file retrieval
    • Use the subspace key to access files under that namespace
    • Use the content hash key to retrieve the file
CHK – Content Hash Key(2)

- Advantages
  - Updating of files
    - Insert an updated version of the file with new content hash key
    - Insert an indirect file (with the old versions name) that stores the location of the new version of the file
    - Key collision will happen when the indirect file reaches a node with the old version of the file
    - Verifies the key, date on file is newer, then replaces it
    - Allows for old files to remain, but will be replaced based on the popularity of the updated version
  - Splitting of files into several pieces
    - Insert each piece with its own content hash key
    - Then use an indirect file to give the locations of each piece of the file
Clustering of Keys

• When a node successfully receives a file from another node
  – It associates that node in its routing table with the hash key of the file
• All future requests from this node will send the request to the node
  – listed in the routing table associated with the key closest to the key of the file being requested
• When an insert is performed
  – The file is passed to the node associated with the closest key
Why Group Files by Hash Key?

- The reason for this design is to spread files on related topics (i.e., P2P) all over the network
  - This prevents one topic from being dependent on a small group of nodes
  - If one node is removed from the network, should not cause most files on one topic, such as P2P, from no longer being accessible
Junk Files

• Keyword signed keys are not very secure
• What if someone wanted to get rid of some file by inserting junk files into the network to take the originals place?
  – Would have the opposite effect
  – Each time a key collision occurs it will see that the new version (junk file) being inserted has the same key as a file that already exists
    • So the junk file will be overwritten by the correct original version
    • Original file is propagated back to the node that inserted the file with the same key, placing a copy on each node the response passes through

• Inserting junk files can result in increasing the numbers of the file that the user is trying to destroy!
Inserting Files

- Insert a new file into Freenet
  - 1. Send an insert message
    - Key (ksk, ssk, chk)
    - # hops
  - 2. Check for file collisions
    - If the local nodes data store has a file with the same key
      - That file is returned as a response, insert is aborted
    - Inserted file is then sent to the node which is associated with similar keys based on the local nodes routing table
      - If a collision on any other node, file is returned, insert is aborted
  - 3. If no key collisions
    - Then # nodes will have copies of the file
Requesting Files

• Availability of files improves over time
• Each time a requestor successfully receives a file from another node
  – 1. It adds that node to its routing table
  – 2. Associates the file key with that node in the routing table
  – 3. All future searches for files with similar keys will be sent to nodes associated with these keys
• Overtime each node should have a better idea who to route a request to based on its routing table
Eventually a node that other nodes associate with a specific key type based on successful requests will:

- Store more files with similar keys
- Reasons:
  - Other nodes send requests for files that have similar keys to that node
  - If it does not have the file it forwards the request to another node based on its routing table
  - When the file is located, the response gets passed back
  - Each node on the responses path gets a copy of the file stored
  - This includes the node the request was initially sent to
  - Over time this node will start to store more and more files with this key type
Standard Request Sequence

Source: “Freenet: A Distributed Anonymous Information Storage and Retrieval System”
http://www.doc.ic.ac.uk/~twh1/academic/papers/icsi-revised.pdf (Page 7)
Response

• No user knows the identity of the node that provides a file they have requested
  – Routing responses through multiple nodes helps to do this
  – Address of previous node is removed after each hop a response message takes
• Each node on the response path gets a copy of the file
  – Helps to increase the number of copies that exist in the network of popular files
  – Makes network less reliant on a single node being connected to make that file available
Announcing Presence

- A new node must announce that it is a part of the network and let other nodes know it is available.
Searching

• You cannot search Freenet in the same manner as other file sharing software
  – Must know the key of a file
    • If file is stored in the network strictly for your own retrieval later on, then you can save the key
    • To allow others to retrieve this file must provide them with the key
Retrieve File Request
Freesites

• Freenet allows users to host web sites
• A subspace key (SSK)
  – Used for access to the web site files
  – SSK@_my_website_private/school
• MapSpace Keys (MSK)
  – Provides accessing of a Freesite based on date
  – Allows users to access older versions of site
    • If they still exist somewhere on the network
• How to create Freenet hosted web sites
Protection of Users and Files

• Users
  – Each user does not know the content of the files they store:
    • All files are stored encrypted
    • Protects users from being prosecuted for the content stored in their data store

• Files
  – Since no user knows the location of every copy of a file and each copy is encrypted:
    • They cannot censor the content by removing a specific file from the network or shutting down one node
  – Since each user can create a subspace that only they have access to and signs each file with a private key:
    • No other user can try to spam the network with fake copies of a file claiming to be from another user
Freenet Summary

• Anonymity is the number one design goal
  – To protect both file providers and requesters

• Freedom of speech protection
  – Prevents censorship by:
    • Encrypting of all content
    • Hiding the locations of nodes that files are being provided from
    • Hiding the identity of file requestors from file providers
    • Allows all users to anonymously insert new files into the network preventing anyone from locating the original source of the file

• Grouping of files based on hash key
  – Prevents network from being dependent on any one node for files all related to one topic

• Increasing copies of files
  – Files that are popular rapidly increase in numbers
    • Responses copy the file to every node the response passes through
Is Freenet Perfect?

• How long will it take to search or insert?
  – Trade off between anonymity and searching efforts: Chord vs Freenet
  – Can we come up a better algorithm? A good try: “Search in Power-Law Networks”

• Have no idea about if search fails due to no such document or just didn’t find it.

• File lifetime. Freenet doesn’t guarantee a document you submit today will exist tomorrow!!