Peer to Peer (P2P) Networking

- What is peer to peer
- P2P structures
- seti@home
- Napster
- Gnutella
- Pastry
- Freenet
- BitTorrent
- JXTA

References

- Chapter 10 Coulouris
- Peer to Peer: Harnessing the Benefits of Disruptive Technologies, ed. By A. Oram, O'Reilly & Associates, 2001
- P2P Networking: An Information Sharing Alternative, M Parameswaran, A. Susarla, A Whinston, IEEE computer, July 2001 pp31-38
- Pastry http://research.microsoft.com/~antr/Pastry/
- BitTorrent: http://www.bittorrent.com/
- Project JXTA www.jxta.org
- www.openp2p.com

What is Peer to Peer Networking?

- A wide-area, resource sharing network for sharing
  - Processing, files, storage
- All nodes are considered 'equal' – as opposed to client-server
- Autodiscovery of peers
- Resources at edge of network (in homes & offices) rather than centralised managed servers.
  - Users contribute resources
- Must cater for intermittent availability of resources
- 'Anti-establishment' philosophy for some of the applications
  - Free music rather than pay for expensive CDs
  - Emphasis on anonymous users to prevent them being traced
- Efficient algorithms needed for data placement across many nodes and subsequent access to data

P2P vs Distributed Processing

Peer to Peer

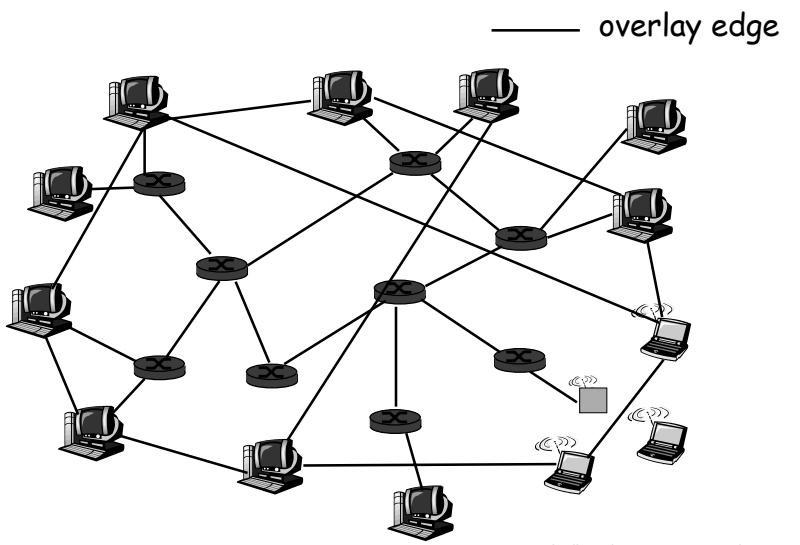
- Millions of nodes cooperate to achieve a common goal
- Distribution of resources are usually explicit but location not known
- WAN based
- Home rather than enterprise based resource servers
- No overall management – insecure resources
- Intermittent connectivity – probabilistic access
- Application level protocols

Distributed Processing

- Smaller numbers of nodes cooperate
- May provide a single virtual machine concept with transparent distribution
- Mostly LAN based
- Within a single or a few enterprises
- Managed system – resources can be more trusted
- Tries to provide deterministic access to resources
- Middleware protocols supporting application level interaction

No fundamental difference

Overlay Networks



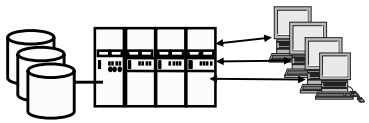
Application Level Routing Overlay

- Link = TCP or IP connection to 'neighbour'
- Neighbours may be chosen based on topology e.g. minimum delay (physical hop count)
- Globally Unique identifiers (GUID) for nodes and stored objects
  - e.g. 128 bit hash of object value by using SHA-1
- Not limited by IP address space – GUID name space > 2^{128}
- Object location randomised and divorced from network topology
- Routing table updates can be synchronous or asynchronous with delays < 1 sec
- Routes and objects can be replicated
- Requests can be routed to find any replica
- Used for DNS, Content distribution networks (CDN), application layer multicast and P2P applications

P2P Structures

- Standard client-server - not P2P

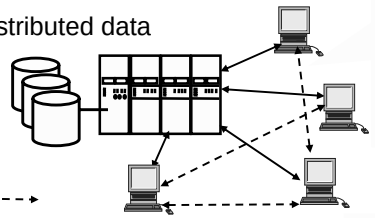
Server maintains data + directory



- Centralised directory, distributed data eg Napster

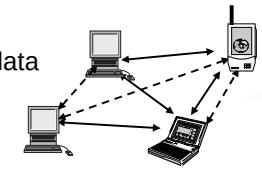
Publish and query to directory. ⟷

Get data from peers ⇠⇢



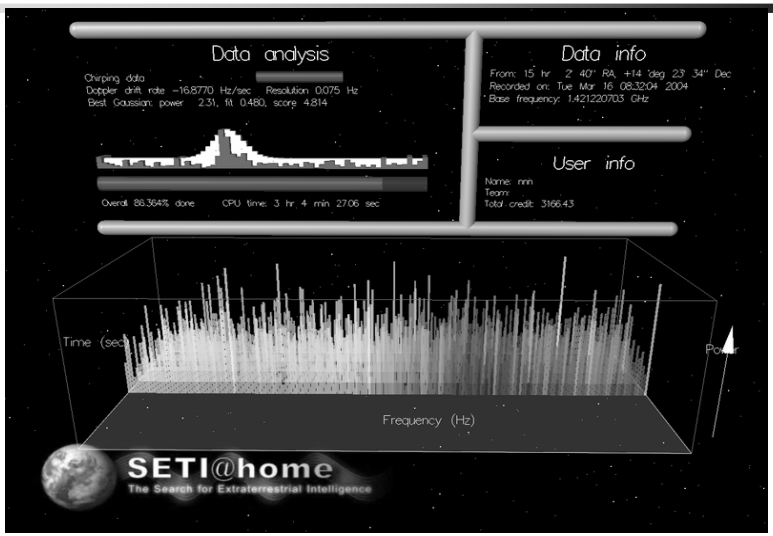
Directory is point of vulnerability – failure, attack, shut down

- Decentralised directory + data eg Gnutella, instant messaging



Where is the directory? Hard to find information

Seti@Home 1



http://setiathome.berkeley.edu/

SETI@home 2

- Sharing of idle processors in home/work machines to analyse signals received from radio telescope
- Works units of about 350Kb distributed redundantly to 3-4 PCs to guard against failures and malicious nodes
- Search for predefined patterns; signals that exceed a threshold are marked and stored in server database
- Download application from centralised SETI server at Berkley
- Request data and return results to server
- > 5.2 m participants

File Lookup Problem

- **Join**: how to begin participating?
  - Broadcast a request
  - Well known site
- **Publish**: how to advertise a file?
  - Centralised/ Replicated server
  - Local content list
- **Search**: how to find a file?
  - Centralised/ Replicated directory
  - Broadcast request
  - Distributed e.g. hash tables
- **Fetch**: how to retrieve a file?
  - Point-to-point FTP
  - Distributed segment download

Napster

- 'Sharing' of music – in reality an application for finding and downloading free MP3 versions of CD tracks
- Centralised, replicated Directory
- Napster provided directory of what users were providing + information on connectivity. Made money from advertising.
- Users provided file storage and bandwidth
- Direct file transfer between users > 60M users
- Napster aided the infringement of copyright laws, although they did not store or copy the music files – eventually shut down

Gnutella 1

- Gnutella is a file sharing network with a decentralised index
- See http://www.gnutelliums.com/
- To join a Gnutella network, locate a suitable node by word of mouth or via various well know servers published on the web.
- Each node maintains a list of other nodes it knows about (neighbours to which it has open TCP connections) - number of connections is a configuration option
- Ping message are used to find out about neighbours and Query messages to search for files
- When a node receives a ping or query message it is multicast to all neighbours, except the one from which it received the message.

Gnutella 2

- Responses to ping and query return along the path by which they were generated.
- Every message has a 16 byte unique ID, and a node discards a message already forwarded. Node remembers message ID
- Node also remembers source from which request was received in order to route response which has same ID as request
- Messages have a maximum Time to live (TTL) count of 7, decremented each time it is forwarded, to limit maximum propagation of multicast.
- When client gets responses to a query it chooses which to retrieve and then retrieves data directly from data source node using an HTTP get.

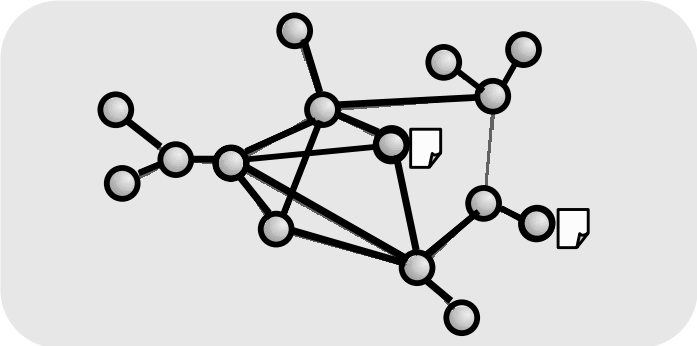
Gnutella Messages

Message UUID	Function	TTL	Hops	Payload length	Payload
16	1	1	1	4	variable

Byte length

- Ping: used to join network and query for neighbours. No payload. Any node receiving a Ping, multicasts it to its neighbours
- Pong: response to a Ping. Payload contains IP address, port, number of files, size of files. A Pong may be returned by nodes receiving the multicast ping.
- Query: used to locate resources. Payload = minimum link data-rate (2), and a variable length search criteria eg file name – can include wildcard characters *
- Query_Hit: response to Query only generated by nodes where the query was successful. Payload = number of hits (1), IP/Port (6), link data-rate,(4), node identifier (16), list of hits consisting of index (4), file size (4). It traverses the reverse path taken by the query

The Gnutella "algorithm"



- Really inefficient
- Graph structure is transient

Gnutella Evaluation

- High data-rate nodes tend to have more connections and act as a backbone
- TTL limits scope of a search and number of visible nodes – gives scalability
- Multicast Ping and Query generate very large traffic. Could use caching from previous results to reduce traffic.
- Gnutella Pseudoanonymity
  - Ping and Query contain no addresses, but Pong and Query-hit do. Routes are only maintained for the time it takes for a query to ripple out and get responses. Very difficult to trace who is searching for files.
  - File store addresses are known and server nodes could log who is accessing file – requires cooperation of server.
  - Needs distributed monitoring – difficult in large network
- Multicast generates high bandwidth usage – very inefficient

Pastry

- Nodes & objects assigned 128 bit GUID computed by applying Secure Hash Algorithm to node's public key or object's name or stored state.
  - GUIDs randomly distributed in range 0 to 2^{128} -1
  - Provide no clue to value from which computed
  - Clashes between GUIDs for different nodes are unlikely
- If GUID identifies node currently active, message delivered to it else to node whose GUID is numerically closest
- Delivery in O(log N) steps
- Routing uses underlying transport to transfer message to node closer to destination (may involve many IP hops)
- Pastry uses locality metric based on hop count or delay in underlying network to select appropriate neighbours when setting up routing tables

Distributed Object Location and Routing in Tapestry

publish(GUID)

GUID can be computed from the object (or some part of it, e.g. its name). This function makes the node performing a *publish* operation the host for the object corresponding to *GUID*.

unpublish(GUID)

Makes the object corresponding to *GUID* inaccessible.

sendToObj(msg, GUID, [n])

Following the object-oriented paradigm, an invocation message is sent to an object in order to access it. This might be a request to open a TCP connection for data transfer or to return a message containing all or part of the object's state. The final optional parameter *[n],* if present, requests the delivery of the same message to n replicas of the object.

Distributed Hash Table API in Pastry

put(GUID, data)

The *data* is stored in replicas at all nodes responsible for the object identified by *GUID*.

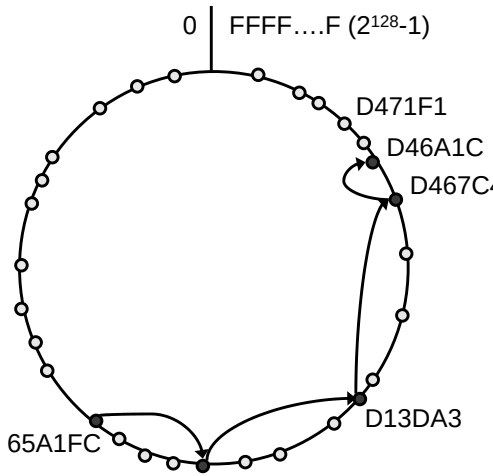
remove(GUID)

Deletes all references to *GUID* and the associated data.

value = get(GUID)

The data associated with *GUID* is retrieved from one of the nodes responsible for it.

Simple Pastry Routing Algorithm



- Each node stores leaf set – vector L (of size 2l) containing GUIDs and IP addresses of l nearest nodes above and l below its GUID
- GUID space is circular: 0's neighbour is 2^{128} -1
- The dots depict live nodes.
- The diagram illustrates the routing of a message from node 65A1FC to D46A1C using leaf set information alone, assuming leaf sets of size 8 (l = 4).
- This is a degenerate type of routing that would scale very poorly; it is not used in practice.

Efficient Pastry Routing Algorithm

- Each Pastry node maintains a tree structured routing table giving GUIDs and IP addresses for a set of nodes spread throughout the entire range of 2^{128} possible values with increased density of coverage for GUIDs numerically close to its own.
- For GUIDs represented as hexadecimal numbers, routing table has as many rows as hex digits in a GUID = 128/4 = 32 rows
- Any row has 15 entries - one for each possible value of the n^{th} hex digit excluding the value in the local node's GUID
- Each entry in table points to one of the potentially *many* nodes whose GUIDs have relevant prefix

First 4 Rows of a Pastry Routing Table

p =	GUID prefixes and corresponding nodehandles n															
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	n	n	n	n	n	n		n	n	n	n	n	n	n	n	n
1	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6F	6E	6F
	n	n	n	n	n		n	n	n	n	n	n	n	n	n	n
2	650	651	652	653	654	655	656	657	658	659	65A	65B	65C	65D	65E	65F
	n	n	n	n	n	n	n	n	n	n		n	n	n	n	n
3	65A0	65A1	65A2	65A3	65A4	65A5	65A6	65A7	65A8	65A9	65AA	65AB	65AC	65AD	65AE	65AF
	n		n	n	n	n	n	n	n	n	n	n	n	n	n	n

The routing table is located at a node whose GUID begins 65A1. Digits are in hexadecimal. **The n's represent** [GUID, IP address] pairs specifying the next hop to be taken by messages addressed to GUIDs that match each given prefix. Grey-shaded entries indicate that the prefix matches the current GUID up to the given value of *p*: the next row down or the leaf set should be examined to find a route. Although there are a maximum of 128 rows in the table, only $\log_{16} N$ rows will be populated on average in a network with *N* active nodes.

Pastry Routing Algorithm

if (destination is within range of our leaf set)
    forward to numerically closest member
else
    if (there is a longer prefix match in table)
        forward to node with longest match
    else
        forward to node in table which
        (a) shares at least as long a prefix
        (b) is numerically closer than this node

New Node Joins

- Compute GUID = SHA (node public key) =X
- Find a nearby pastry node & measure round trip delay to all its leaf nodes. Choose one with lowest delay as nearest neighbour = A.
- Send join (X) to A. A routes this to Z which is numerically closest to X.
- A, Z and any intermediate nodes (B, C, D etc) via which the message has passed, send the relevant parts of their routing tables and leaf node set to X.
- X constructs routing table:
  - First row = A's first row but A & X will probably have different first digit
  - B has same first digit as X so use it's second row, similarly 3rd row from C.
  - Use Z's leaf set, which should differ by only 1 member from Xs so can be used for initial value of leaf set.
- X sends its routing table and leaf set to all nodes in the leaf set

Locality Selection

- Pastry routing is highly redundant with many routes between a pair of nodes.
- Entry in row *i* gives 16 nodes with *i*-1 initial hex digits that match the node's GUID. Received information from other nodes should give more than 16 candidate entries. Measure delay to candidates and choose ones with lowest delay (or hop count).
- Does not produce optimal routing table but simulations show only 30-50% longer than optimum.

Node Leaves or Fails

- Node fails when immediate GUID neighbours cannot communicate with it.
- Node detecting failure in its leaf set, finds another node closest to failed one, and requests a copy of its leaf set. This will contain a partial overlap with the requesting node's leaf set, so it finds one with a suitable value to replace the failed node.
- Reports failure to all neighbours in leaf set which perform similar update
- Guarantees leaf set repair unless all fail simultaneously

- Nodes send heartbeat messages at regular intervals to neighbouring nodes in leaf set – can be used to detect failure
- Reliable message forwarding using timeouts and retransmissions also detects failures of nodes. Failed nodes are replaced in table.
- For randomly selected small proportion of cases, route to nodes with common prefix less than maximum length – bypasses malicious nodes giving incorrect routing information

Pastry Performance

- Simulation and Implementation on network of 52 nodes
- Loss of 1.5 in 100K messages assuming no IP losses due to non availability of destination
- Loss of 3.3 in 100K messages with IP loss rate of 5%
- Delay increase in delivery time compared to normal UDP/IP delivery = 80% for zero IP loss and 120% for 5% network IP loss
- Overheads due to control and update traffic < 2 message per minute although much more for short sessions.

Squirrel Web Cache Using Pastry

- Browsers can use centralised proxies to cache frequently accessed web pages. Squirrel uses small part of resources of client workstations to do the same.
- SHA function applied to URL to produce 128 bit GUID. Node with closest GUID becomes object's home node and caches copy of the object.
- Clients have a local Squirrel proxy process which manages local cache. If object is not in local cache a get is sent to home node, which may return a fresh copy or request one from the origin server
- Evaluation showed 30-38% hit ratio but delay overheads in LAN based systems make it too slow. However works well for WAN based server access. Very low proportion of system resources used: 0.31 requests per min average per node.

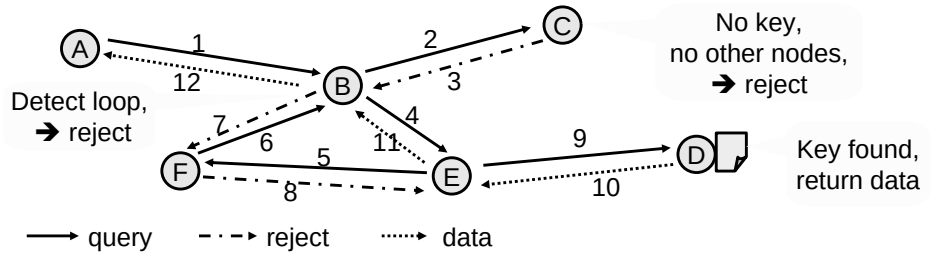
Freenet

- Used to store documents and offers anonymity to both publishers and clients accessing documents for *freedom of information*
- Anonymous and survivable information storage
- Clients must offer storage as well as using Freenet to store files
- Redundant storage prevents loss of data due to attacks or failure
- Files have a globally unique ID key (16bytes) – used for routing to find file.
- Files may be encrypted using different key
- Privacy
  - Messages routed via chains not directly from source to recipient
  - Cannot tell whether neighbour in chain is intermediate or recipient/source.
  - Messages are encrypted and padded to standard length
- See http://freenet.sourceforge.net/

Freenet Requesting Files

- Node maintains routing table of subset of other nodes with keys it thinks it holds.
- Node receives query and first checks own store for key. If found returns file with tag identifying itself as source. If not found it looks up numerically-closest key in its table and forwards request. This node repeats actions. If request eventually succeeds, the file is sent back along the reverse of the query path. Nodes along the path may cache a copy depending on its distance from the source.
- Queries have a TTL count, decremented at each node. If it reaches zero the query fails and an error message is returned. If a query loops back it is rejected, and the sender tries the next-closest key. If all keys fail it reports failure to its predecessor in the query chain which tries its next closest key and so on.
- Requests home in closer until key is found. Subsequent queries will follow the path taken by first query but may be satisfied by a cached copy along the way & queries for similar keys will also go to nodes which have successfully supplied data.

Freenet - Example Query



- Nodes that reliably answer queries gain routing table entries along all nodes in the chain and are contacted more often than nodes that do not
- Graph structure adaptively evolves over time
  - new links form between nodes
  - files migrate through network
- Anonymity through request chains

Freenet - File Insert

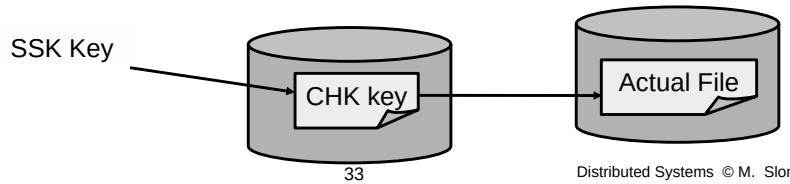
- User assigns GUID Key and sends insert message with Key and TTL field indicating number of copies to be stored.
- Node receiving insert checks if key already exists, if so returns files and insert fails, so user must choose another key and try again. If key not found, then node looks up closest key and forwards insert to corresponding node as for queries.
- If an insert fails with a file returned, it is treated as the return from a query. Routing table is updated, file may be cached and forwarded upstream.
- If TTL expires, without collision, an all-clear is returned upstream along the reverse path of the insert. The user then send the file along the same path as the initial insert. Each node along the path updates its routing table setting the source of the data as the furthest downstream node in the chain where the insert message exceeded TTL.
- An insert follows the same path, updates routing tables, and stores files in the same nodes as a successful query.
- Inserts of similar keys follow the same paths so similar keys cluster in nodes in those paths

Signed Subspace Key

- Signed Subspacekey (SSK): personal namespace which can only be written by owner.
- User creates a subspace by generating a public/private key pair to identify it.
- Choose a short text description eg politics/us/bombings.
- Hash public key. Hash text description, concatenate and hash again ➔ SSK ie SSK = H(H(text) + H(Kp))
- Private key used to sign file
- To retrieve file you need public key and text description to generate SSK
- Adding or updating file needs private key ➔ all files in a signed subspace are generated by same person ie owner of private key.

Content Hash Key (CHK)

- CHK = Hash (file contents)
- Unique identifier, easy to authenticate.
- Can easily identify identical copies of files – same CHK
- Need CHK to retrieve file ➔ not easy to retrieve files inserted by others.
- Combine SSK and CHK using indirect reference
  - Insert file using CHK
  - Insert indirect file containing CHK using a text description under SSK
  - Others can retrieve file knowing public key and text description
  - Original file can be updated which results in a new CHK which must be used for updating indirect file under SSK

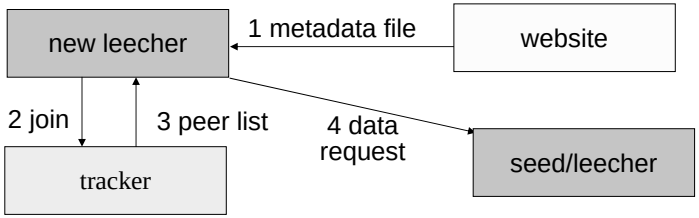


BitTorrent

- Multimedia Content Distribution
  - Large files such as movies, clips etc. - 25% internet traffic!
- Focused on efficient *fetching*, not *searching*
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
  - Downloaders share downloaded segments
- Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - CNN on 9/11, new movie/game release
  - Slashdot effect – popular website links to a smaller website within a news story
- Employ "Tit-for-tat" sharing strategy
  - "I'll share with you if you share with me"
  - Be optimistic: occasionally let freeloaders download
  - Necessary for starting download process

BitTorrent Overview

1. Locate metadatafile called file.torrent e.g. via Google
   - Contains length, name, hash, URL of tracker
2. Query Tracker – join a torrent
3. Tracker provides randomly selected list of peers downloading:
   - Seeds: have entire file
   - Leechers: still downloading
4. Contact peers from list to request data
5. Peers form P2P network



BitTorrent Pieces

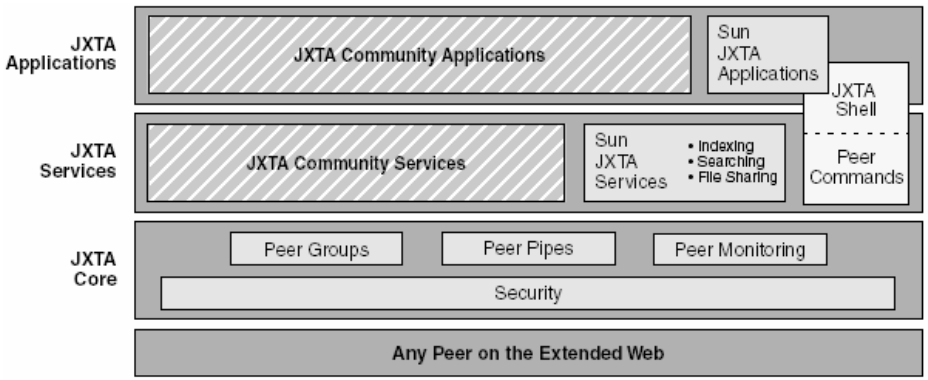
- File is broken into pieces
  - Typically piece is 256 Kbytes – can have 16Kbyte sub-pieces
  - Download pieces in parallel
  - Advertise received pieces to peer list
  - Upload pieces while downloading pieces
- Piece selection
  - At download start, select random pieces
  - Select rarest piece, so that available to others
- Upload (Unchoke) Selection
  - Periodically calculate download rate
  - Select up to 4 peers for uploading that download at the highest rates
- Optimistic Upload
  - Periodically (30sec) select a peer at random and upload to it
  - Continuously look for the fastest partners

BitTorrent Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources
  - Avoids freeloaders
- Cons:
  - Central tracker server needed to bootstrap swarm

JXTA Tools

- Java based toolset for implementing P2P applications
- See www.jxta.org



JXTA Technology

- Advertisement: XML description of peer, group, pipe or service
- Asynchronous messages over UDP
- Pipes are asynchronous, unidirectional channels for sending & receiving messages
  - Point-to-point connecting 2 peer endpoints
  - Propagate: one-to-many
  - Pipes can be bound to different peers at different times cf Unix pipes
- Peer Discovery – finds advertisements via multicasts, references in messages, cascaded via intermediaries, using rendezvous points (peers holding information on other peers eg web site)
- Peer resolver: searches peers which have data repositories
- Security via crypto library, authentication, peer group access control , SSL

Summary

- P2P is essentially distributed resource sharing + hype
- Home rather than enterprise based resources
- Large-scale resource searching and download strategy are the most interesting aspects
- Initial emphasis on ‘anti-establishment’ applications eg free MP3 music and freedom of information
- Recent interest in commercial applications
  - Parallel processing using idle workstation within enterprise or on the net eg United devices Grid solution
  - Replicating file stores eg OceanStore
  - Distributed content management – searchable data distributed on user machines
  - Instant messaging and VOIP
  - Collaboration based on ad hoc groups eg Groove