Modern Techniques on Improving the GNet

J. L. Chiang

MNet Lab.

Dec. 2, 2004

Outlines

- Connecting GNet
- Gnutella messages
- Pong caching
- Flow control
- Ultrapeer system
- Query Routing Protocol
- Emerging P2P related products
- Gnutella2

Terminologies

- Servent (SERVer+cliENT)
  - Client, servant
  - Peer, node, host
- Message
  - Packet, descriptor
- GUID (Globally Unique Identifier): 16-byte
- Gnutella Network (GNet)
- Gnutella Development Forum (GDF)
  - http://groups.yahoo.com/group/the_gdf

Bootstrapping

- Launch the client and load the **host cache**.
- Try to connect to the GNet, using the host cache.
- If after X seconds, there's no connection to the GNet, query a random **GWebCache**.
- Wait for at least Y seconds before sending a request to another GWC (it may happen that the first chosen GWC is down, or very slow to answer).
- After Y seconds, if no results have been received from the first GWC call, call a new GWC each Z seconds, until a response is received from one of them.
- Once results have been received from (at least) one GWC, the servent SHOULD NOT send a new request during the next session, unless the host cache is empty (which should not happen if there is an initial host cache provided with the software and no option to clear the host cache is available).
- Suggested values in seconds for X, Y and Z are respectively 5, 10, 2.

Host Cache

- In order to help the bootstrap process, a servent SHOULD implement a hosts cache, keeping **a few hundreds** hosts on the disc.
- It is filled with hosts grabbed from **GWebCache** calls, from X-Try and **X-Try** headers, from **Pongs**, and optionally from **QueryHits**.
- A **starting hosts list** SHOULD be distributed with the client, filled with known servents having a regular activity on the GNet, so that even the very first time the servent is launched, it will be able to use the host cache.

GWebCache (GWC)

- The cache is a program (**script**) placed on any **web server** that stores IP addresses of hosts in the Gnutella network and URLs of other caches.
- Gnutella clients connect to a cache in their list randomly. They send and receive IP addresses and URLs from the cache.
- With the randomized connection it is to be assured that all caches eventually learn about each other, and that all caches have relatively fresh hosts and URLs.
- The concept is **independent** from Gnutella clients.

Initiating a Connection

```
Client                                  Server
-------------------------------------------------------------
GNUTELLA CONNECT/0.6
User-Agent: BearShare/1.0
Pong-Caching: 0.1
GGEP: 0.5

                                        GNUTELLA/0.6 200 OK
                                        User-Agent: BearShare/1.0
                                        Pong-Caching: 0.1
                                        GGEP: 0.5
                                        Private-Data: 5ef89a

GNUTELLA/0.6 200 OK
Private-Data: a04fce


[binary messages]                       [binary messages]
```

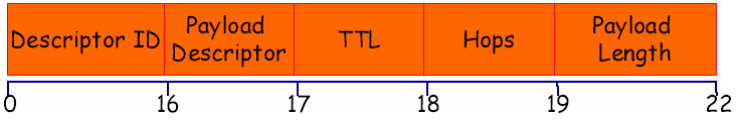
- The X-Try header is not shown here.

X-Try Header

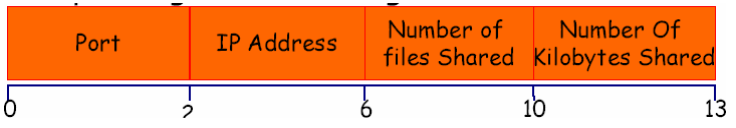
- Every servent SHOULD send an X-Try header during the handshake.
- This header gives a list of hosts to which the servent can attempt to connect, allowing it to get new host addresses without using the GWebCache.
  - X-Try: 1.2.3.4:1234, 1.2.3.5:6346, 1.2.3.6:6347
- A servent SHOULD send a reasonable number of hosts, common values are **between 10 and 20**.

Gnutella Messages

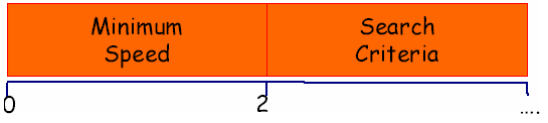
- Descriptor
- Ping (0x00)
  - Used to actively discover hosts on the network. A servent receiving a Ping message is expected to respond with one or more Pong messages.



- Pong (0x01)
  - The response to a Ping. Includes the address of a connected Gnutella servent, the listening port of that servent, and information regarding the amount of data it is making available to the network.

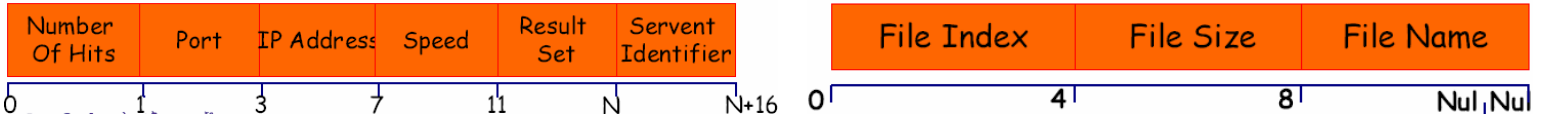


- Query (0x80)
  - The primary mechanism for searching the distributed network. A servent receiving a Query message will respond with a Query Hit if a match is found against its local data set.

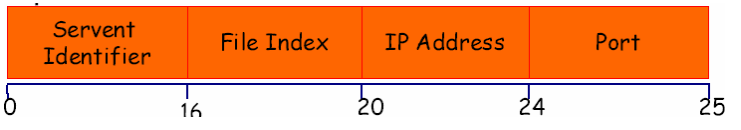


Gnutella Messages

- QueryHit (0x81)
  - The response to a Query. This message provides the recipient with enough information to acquire the data matching the corresponding Query.



- Push (0x40)
  - A mechanism that allows a firewalled servent to contribute file-based data to the network.



- Bye (0x02)
  - An optional message used to inform the remote host that you are closing the connection, and your reason for doing so.

GGEP

- The Gnutella Generic Extension Protocol (GGEP) allows arbitrary extensions in Gnutella messages.
- Flags:
  - bit7: Last Extension.
  - bit6: Encoding.
  - bit5: Compression.
  - bit4: Reserved.
  - bits3-0: ID Len Value 1-15 can be stored here.
- ID: The raw binary data in this field is the **extension ID**.
- Data Length: This is the length of the raw extension data.
- Extension Data: The actual extension data.

Pong Caching

- Pong caching **reduces bandwidth consumption** in the Ping-Pong network discovery phase.
- For **each connection** an array of Pong messages are stored.
- When a Pong comes in, it overwrites the oldest stored Pong in array of the connection the Pong came from.
- The information that must be stored for each Pong is:
  - IP Address
  - Port number
  - Number of files shared
  - Number of kilobytes shared
  - GGEP extension block (if present)
  - Hops value

Pong Caching

- When a Ping message is received (TTL>1 and it was at least one second since another Ping was received on that connection), a servent responds with a number of (e.g. 10) Pong messages.
- Incoming pings with **TTL=1** and **Hops=0 or 1** is replied to with a single pong containing information about the local host.
- Pings with **TTL=2** and **Hops=0** (Crawler Ping) are replied to with one pong about the local host, and one about each other host the local host is connected to.

Search Criteria (Query)

- The Search Criteria is **a string of keywords**.
- A space is the standard separator between words.
- Regular expressions are not supported and common regular expression "meta- characters" such as "*" or "." will either stand for themselves or be ignored.
- The matching SHOULD be **case insensitive**.
- Empty queries or queries containing only 1-letter words SHOULD be ignored.
- Servents MAY ignore queries whose Search Criteria is **shorter than a chosen length** to ignore too broad searches.
- Query messages with TTL=1, hops=0 and Search Criteria=“ “ (**four spaces**) are used to index all files a host is sharing.

Flags Field Semantics (Query)

- bit15: Flags field semantics.
- bit14: Firewalled indicator.
- bit13: XML Metadata.
- bit12: Leaf Guided Dynamic Query.
- bit11: GGEP "H" allowed.
- bit10: Out of Band Query.
- bit9: Reserved for a future use.
- bits0-8: Indicates the maximum number of query hits expected, 0 if no maximum.

Sending and Forwarding Query Messages

- Query messages are usually sent when the user initiates a search.
- A servent MAY also create Queries automatically, to find more locations of a resource for example. If doing so the servent MUST be very careful not overload the network. A servent SHOULD not send more than one **automatic query** per hour.
- Servents SHOULD NOT allow the user to create a large amount of queries by **repeatedly clicking** on a button.
- Servents SHOULD watch queries originating from its neighbors (Hops=0) If those queries are too frequent, are duplicates or indicate **bad servents behavior** in any other way, the servents SHOULD drop those queries or even close the connection.
- The TTL value of a new query created by a servent SHOULD NOT be higher than 7, and MUST NOT be higher than 10. The hops value MUST be set to 0.

EQHD (QueryHit)

- x: RECOMMENDED extra block (EQHD or QHD).
  - Bytes0-3: Vendor Code.
  - Byte4: Open Data Size.
  - X: Open Data.
    - bits7,6: Reserved for future use
    - Bit5: flagGGEP
    - Bit4: flagUploadSpeed
    - Bit3: flagHaveUploaded
    - Bit2: flagBusy
    - Bit1: Reserved for future use
    - Bit0: flagPush
- x: Private data

Bye

- Bytes0-1: Code.
  - 2xx The User did nothing wrong, but the servent chose to close the connection: it is either exiting normally (200), or the local manager of the servent requested an explicit close of the connection (201).
  - 4xx The User did something wrong, as far as the servent is concerned. It can send packets deemed too big (400), too many duplicate messages (401), relay improper queries (402), relay messages deemed excessively long- lived [hop+TTL > max] (403), send too many unknown messages (404), the node reached its inactivity timeout (405), it failed to reply to a ping with TTL=1 (406), or it is not sharing enough (407).
  - 5xx The servent noticed an error, but it is an "internal" one. It can be an I/O error or other bad error (500), a protocol desynchronization (501), the send queue became full (502).
- Bytes2-x: NULL-terminated Description String.

Flow Control

- Implement an **output queue**, listing pending outgoing messages in FIFO order.
- As long as the queue is less than, say, 25% of its max size (in bytes queued, not in amount of messages), do nothing.
- If the queue gets filled above 50%, enter flow-control (FC) mode.
- In FC mode, all **incoming queries** on the connection are dropped. The rationale is that we would not want to queue back potentially large results for this connection since it has a throughput problem.

Message Priority

- Messages to be sent to the node (i.e. queued on the output queue) are prioritized:
  - For broadcasted messages, the more hops the packet has traveled, the less priority it is. **Or** the less hops, the more priority. This means your own queries are the most priority (hops = 0).
  - For replies (query hits), the more hops the packet has traveled, the more priority it is. This is to maximize network **usefulness**. The packet was relayed by many hosts, so it should not be dropped or the bandwidth it used would become truly wasted.
  - Individual messages are prioritized thusly, from the most priority to the least: Push, Query Hit, Pong, Query, Ping. The **Bye** message being special, it is always sent (i.e. the queue cannot be in FC mode since it needs to be cleared before sending Bye).

Ultrapeer System

- Originally, all Gnutella nodes were connected to each other randomly. It worked fine for users with broadband connections, but not for users with slow **modems**. (66%)
- Ultrapeers are connected to each other and to "normal" Gnutella hosts. Ultrapeers maintain many (10-100) leaf connections, as well as a small (<10) number of connections to other ultrapeers.
- A leaf keeps only a small number of connections open, and that is to ultrapeers. Leaves MUST NOT connect to more than 3 ultrapeers by default. Leaves SHOULD make it difficult for the user to adjust these defaults and MUST NOT ever allow the user to connect to more than 10 ultrapeers.

Ultrapeer System

- An ultrapeer acts as a proxy to the Gnutella network for the leaves connected to it.
- Ultrapeers **shield** leaf nodes from virtually all ping and query traffic in one of two ways:
  - **Reflector indexing**: Ultrapeers periodically send an indexing query to leaf nodes. Leaf nodes respond with a query reply naming all shared files. The ultrapeer uses these replies to build an index of its leaves' contents. The ultrapeer then responds to queries on behalf of leaf nodes, shielding them from all query traffic.
  - **QRP routing**: leaf nodes periodically send ultrapeers query routing tables using LimeWire's proposed Query Routing Protocol (QRP). Ultrapeers then forward queries only to those leaf nodes whose QRP table has a corresponding entry. Leaves respond in the normal manner. Ultrapeers do not propogate QRP tables amongst themselves.

QRP

- At the leaf node level:
  - Break all the resource names into **individual words**. A word is made of a consecutive sequence of letters and digits.
  - Hash each word with a well-known **hash function** and insert a "present" flag in the corresponding hash table slot. Note that this hash table is a big array, and we don't store the key, only the fact that a key ended up filling some slot. All words are lower-cased and all accents are removed from them, so that only **ASCII** characters remain. Only those words that are made of **at least 3 letters** are retained.
  - All words are re-hashed with their trailing 1, 2, or 3 letters removed, provided the word length after such trimming is at least 3 letter long. This is a simple attempt to **remove plural from words**. Optionally, nodes can chop off more letters from the end, provided that each hashed word is at least 3 character long.
  - The "**boolean vector**" built at later stage is optionally compressed, broken up in small messages, and sent mixed with regular GNet traffic to the ultrapeer.

QRP

- At the Ultrapeer level:
  - Until the whole "boolean vector" is received from a leaf node, all queries are forwarded to that node.
  - When the "boolean vector" is fully received, it is going to be used as the **Query Routing table** for that leaf node: queries are broken into individual words, all accentuated letters are removed.
  - For each leaf node with a Query Routing table:
    - Each word is then hashed and looked up in the Query Routing table.
    - Depending on the query matching rules, either ALL the words will be required to be found in the Query Routing, or only some of them, to declare a **Query Routing Hit**.
    - Only those queries that were declared a Hit at the previous stage will be forwarded to a given leaf node.

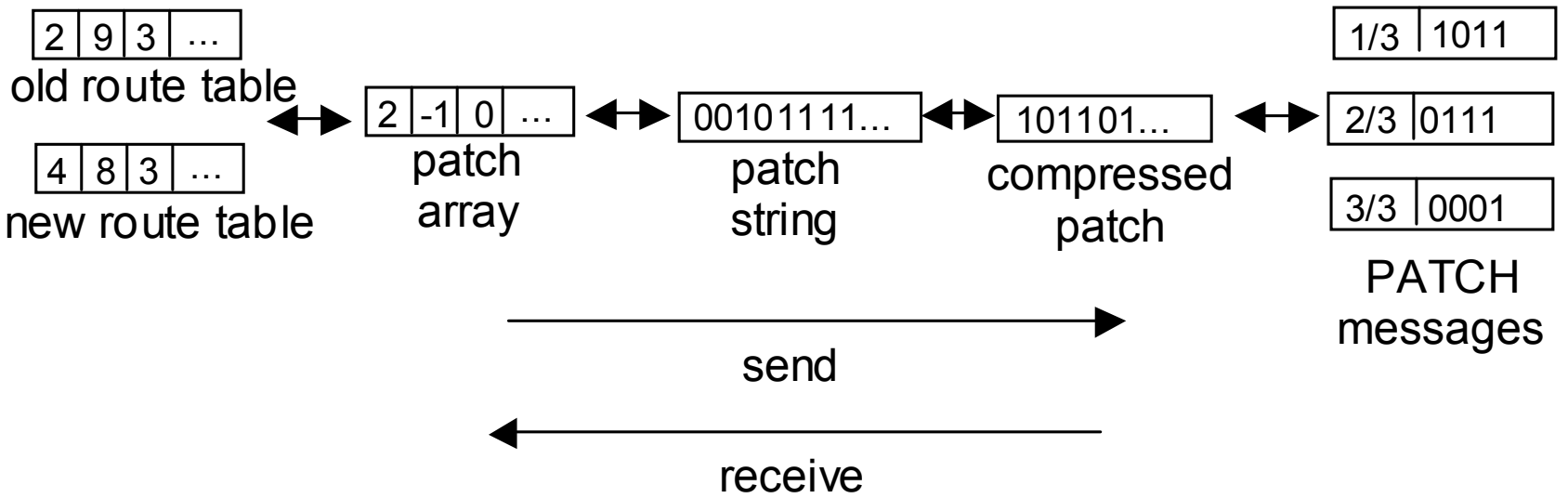
Adv's of Using QRP

- **Better QHD support**: because leaf nodes are given the chance to respond to all queries, they can provide **up-to-the-minute** information in the QHD, such as estimated download speed and busy status.
- **Update efficiency**: because QRP tables eliminate redundant keywords and add compression, they are likely to be significantly smaller than replies to indexing queries. Furthermore, a leaf node only needs to send **incremental QRP updates** when its shared files have changed. In contrast, a reflector-style ultrapeer would need to periodically re-index all of the leaves' files—this takes more bandwidth.

Adv's of Using QRP

- **CPU efficiency**: it is very cheap (constant time) for a ultrapeer to decide whether to forward a query to a leaf node with the QRP proposal. The reflector-style scheme can be implemented efficiently, but it is considerably more difficult.
- **Privacy**: ultrapeers do not actually know what files are shared by leaf nodes—only those files' hashes.
- **Ease of implementation**: LimeWire has already implemented QRP, and the code is available to the public under the GNU Public License (GPL). Maintaing a single QRP table per connection is easier to implement than building a "virtual file manager" for all connections.

Incremental QRP Updates



Ultrapeer Election

- Not firewalled.
- Suitable operating system.
- Sufficient bandwidth.
- Sufficient uptime.
- Sufficient RAM and CPU speed.

- If the above criteria are met, a node is said to be **ultrapeer capable**.
- Whether an ultrapeer capable node will actually become an ultrapeer depends on if there is need for more ultrapeers on the network, and on how well the above criteria are met.

Ultrapeer Handshaking

- X-Ultrapeer:
  - "True" signals that node is an ultrapeer, "False" signals that the node wants to be a shielded leaf node.
- X-Ultrapeer-Needed:
  - Used to balance the number of ultrapeers.
- X-Try-Ultrapeers:
  - Like X-Try, but contains only addresses of ultrapeers.
- X-Query-Routing:
  - Signals support for the Query Routing Protocol. The header value is the QRP version (curretly 0.1).

Leaf to Ultrapeer

```
Leaf                                Ultrapeer
-------------------------------------------------------------
GNUTELLA CONNECT/0.6
User-Agent: LimeWire/1.0
X-Ultrapeer: False
X-Query-Routing: 0.1

                                    GNUTELLA/0.6 200 OK
                                    User-Agent: LimeWire/1.0
                                    X-Ultrapeer: True
                                    X-Ultrapeer-Needed: False
                                    X-Query-Routing: 0.1
                                    X-Try: 24.37.144:6346,
                                     193.205.63.22:6346
                                    X-Try-Ultrapeers: 23.35.1.7:6346,
                                     18.207.63.25:6347

GNUTELLA/0.6 200 OK


[binary messages]                   [binary messages]
```

Leaf to Leaf

- Case 1:

```
Leaf1                                   Leaf2
-------------------------------------------------------------
GNUTELLA CONNECT/0.6
X-Ultrapeer: False

                                        GNUTELLA/0.6 503 I am a leaf
                                        X-Ultrapeer: False
                                        X-Try: 24.37.144:6346
                                        X-Try-Ultrapeers: 23.35.1.7:6346


                                        [Terminates connection]
```

- Case 2:

```
Leaf1                                   Leaf2
-------------------------------------------------------------
GNUTELLA CONNECT/0.6
X-Ultrapeer: False

                                        GNUTELLA/0.6 200 OK
                                        X-Ultrapeer: False

GNUTELLA/0.6 200 OK


[binary messages]                       [binary messages]
```

Ultrapeer to Ultrapeer

- Case 1:

```
UltrapeerA                          UltrapeerB
-----------------------------------------------------
GNUTELLA CONNECT/0.6
X-Ultrapeer: True

                                    GNUTELLA/0.6 200 OK
                                    X-Ultrapeer: True


GNUTELLA/0.6 200 OK


[binary messages]                   [binary messages]
```

- Case 2:

```
UltrapeerA                          UltrapeerB
-----------------------------------------------------
GNUTELLA CONNECT/0.6
X-Ultrapeer: True

                                    GNUTELLA/0.6 200 OK
                                    X-Ultrapeer: True
                                    X-Ultrapeer-Needed: False


GNUTELLA/0.6 200 OK
X-Ultrapeer: False


[binary messages]                   [binary messages]
```

File Transfer

- Result Set:
  - File Index: 2468
  - File Size: 4356789
  - File Name: Foobar.mp3
- HTTP download request:
  - GET /get/2468/Foobar.mp3 HTTP/1.1
  - User-Agent: Gnutella
  - Host: 123.123.123.123:6346
  - Connection: Keep-Alive
  - Range: bytes=0-
- Reply:
  - HTTP/1.1 200 OK
  - Server: Gnutella
  - Content-type: application/binary
  - Content-length: 4356789
  - DATA…

File Transfer

- The most important features for Gnutella, range requests and Persistent Connections MUST be supported.

- Range request
  - GET /get/2468/Foobar.mp3 HTTP/1.1
  - User-Agent: Gnutella
  - Host: 123.123.123.123:6346
  - Connection: Keep-Alive
  - Range: bytes=4932766-5066083

- Reply
  - HTTP/1.1 206 Partial Content
  - Server: Gnutella
  - Content-Type: audio/mpeg
  - Content-Length: 133318
  - Content-Range: bytes 4932766-5066083/5332732
  - DATA…

Firewalled Servents

- If a direct connection cannot be established, the servent attempting the file download may request that the servent sharing the file "**push**" the file instead.
- A servent can request a file push by routing a Push request back to the servent that sent the QueryHit message describing the target file.
- The servent that is the target of the Push request (identified by the Servent Identifier field of the Push message) SHOULD, upon receipt of the Push message, attempt to establish a new TCP/IP connection to the requesting servent.
  - GIV <File Index>:<Servent Identifier>/<File Name><cr><lf>
- The servent that sent the GIV MUST allow the client to request **any file**, and not just the one specified in the Push message.

Busy Servents & Upload Queueing

- Servents whose upload bandwidth is already saturated with transfers MAY **reject** a download request by returning the 503 response code.
- If a transfer is interrupted, the serving servent SHOULD keep the allocated slot/bandwidth reserved for at least one minute. The downloader would then be allowed to reconnect and **resume** the transfer.
- Clients which support queues send "X-Queue: 0.1", which simply tags the request as a candidate for queuing
  - X-Queue: position=2,length=5,limit=4,pollMin=45,pollMax=120
- Upload queues represent an important step for the evolution of Gnutella because they reward users who have waited for a file, rather than a "luck of the draw" approach which (if anything) rewards users who abuse the system by requesting too often, etc.

Active Queuing Extension

- The downloader can see their place in the queue change as they move towards position #1, so even if the queue is long, at least **progress** can be observed.
- Because the HTTP request is reissued **periodically**, the client is able to request the most appropriate "Range" each time.
- By requiring the requesting client to maintain a connection, there is no need to hold open upload positions for a request that may never come.
- The pollMin and pollMax time variables can be made proportional to the position within the queue.
- Query-hit output MAY be **adjusted** based on the state of the upload queues.

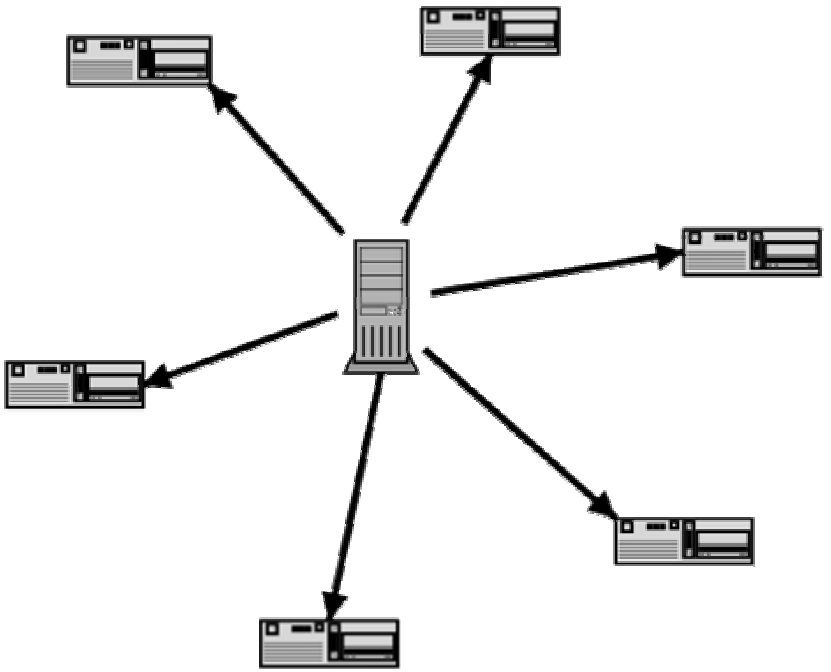
Sharing

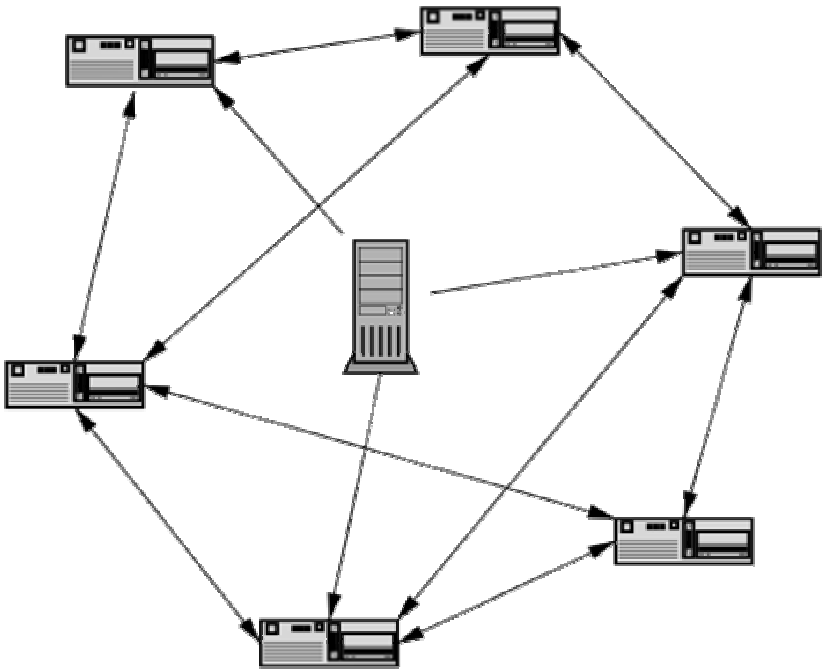
- Servents that are able to download files MUST also be able to share files with others.
- Servents SHOULD encourage users to share files.
- Servents SHOULD, by default, share the directory where downloaded files are placed.
- Servents SHOULD also share new downloaded files without waiting for the servent to be restarted.
- Servents SHOULD avoid changing the index numbers of shared files.
- Partial files MAY be shared in a way the makes it clear to other servents that the file is incomplete.

BitTorrent Overview

- BitTorrent is a protocol for distributing files.
- It identifies content by **URL** and is designed to integrate **seamlessly** with the web.
- Its advantage over plain HTTP is that when multiple downloads of the same file happen concurrently, the downloaders upload to each other, making it possible for the file source to support very large numbers of downloaders with only a modest increase in its load.

BitTorrent Overview

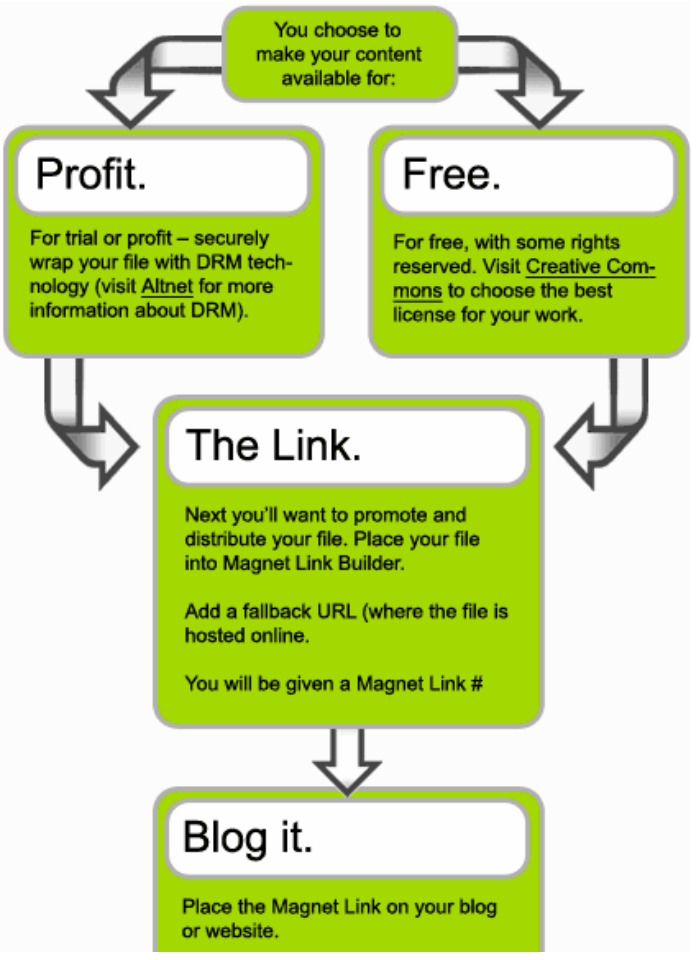




Magnet Overview

- Magnet links allow users to directly download large media files saving website creators and bloggers money on bandwidth costs and effectively propagating files on p2p networks that attract millions of users per day.
- They are supported by the most popular p2p applications including: Kazaa Media Desktop, Limewire, Morpheus, Shareaza, Bearshare, Xolox.
- Magnet links can also be used to initiate searches.

Magnet in KaZaA

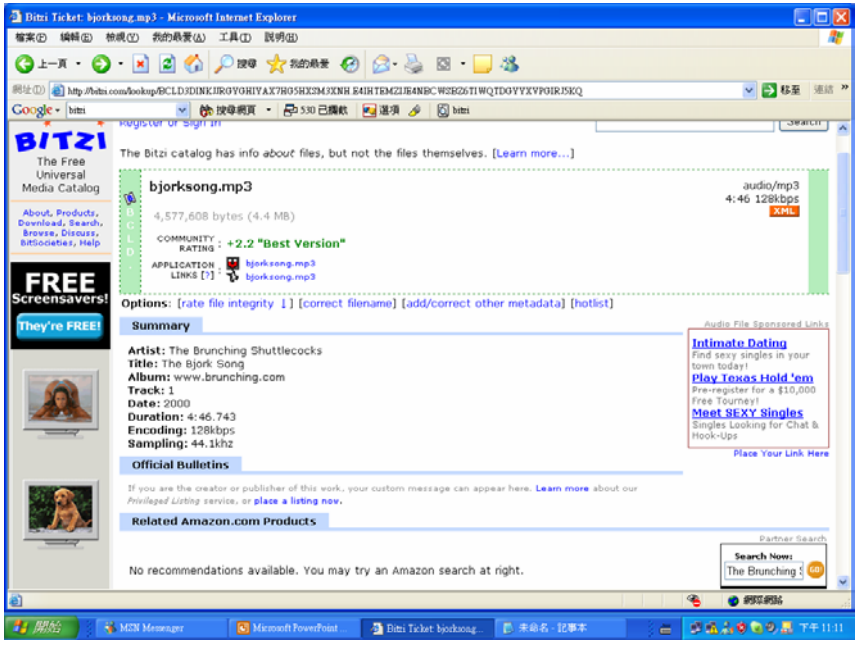


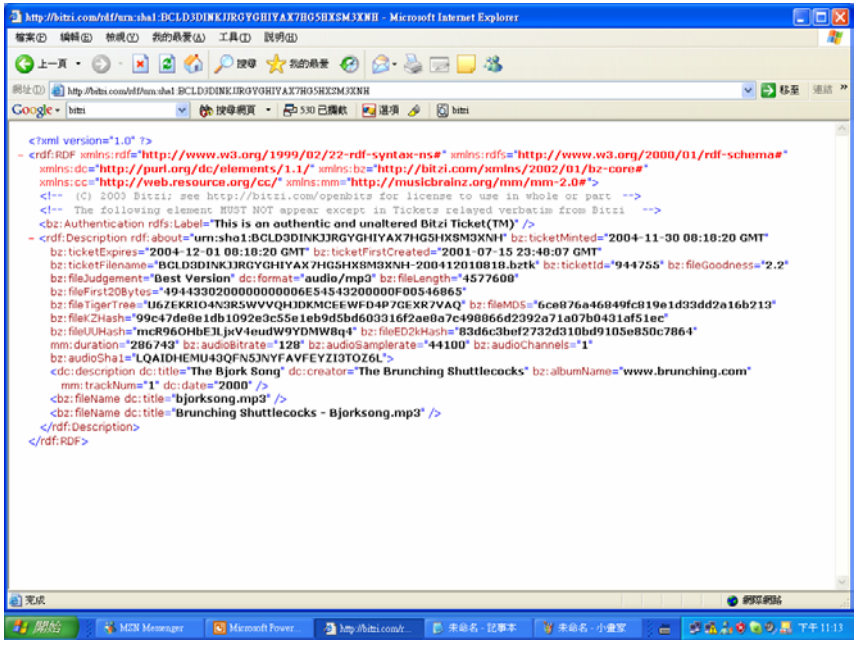
Bitzi Overview

- Bitzi is a website where people cooperate to identify, describe, and discover files of all types.
- With Bitzi:
  - You can look up descriptions, comments, and ratings about your files – or contribute such info yourself.
  - Our precise digital **fingerprints** match info to exact files, so you can distinguish between similar files and search for the very best versions.
  - File-sharing tools can assure you of a file's contents before you begin downloading.
  - Infected or mislabeled files can be flagged, and so discovered or ignored before doing any harm.
- The Bitzi catalog is an open resource built by a community of fans, developers, and creators.

Bitprint

- http://bitzi.com/lookup/»bitprint«
- http://bitzi.com/lookup/BCLD3DINKJJRGYGHIYAX7HG5HXSM3XN H.E4IHTEMZIJE4NBCWSBZ6TIWQTDGYYXVPGIRJ5KQ





G2

- Gnutella2 is a modern and efficient peer-to-peer network standard and architecture designed to provide a solid foundation for **distributed global services** such as person to person communication, data location and transfer and other future services.
- The **Gnutella2 Network** is perhaps the most easily recognised component. It is a new high-performance peer to peer network architecture upon which a variety of distributed applications can be built, such as file sharing applications, communication tools, etc.
- The **Gnutella2 Standard** is a set of requirements for building applications which operate on the Gnutella2 network in different capacities. It specifies the minimum compliance level required to be recognised as a **Gnutella2-compatible** application. Compliance with a Gnutella2 Standard ensures participating applications provide a minimum acceptable level of service to other network participants.

Common Gnutella2 Standard

- Bidirectional TCP stream connections (stream compression OPTIONAL)
- Bidirectional reliable UDP protocol (Gnutella2 reliability layer and stateless compression REQUIRED)
- HTTP-style link negotiation, exchanging at least the required headers
- Gnutella2 protocol support, graceful handling of unknown trees
- Localised, UTF-8 and UNICODE decode REQUIRED, encoding to each optional
- Operation in LEAF mode, additional node states OPTIONAL
- Basic link handshaking and maintenance functionality (PI/PO/LNI/KHL)
- Global node addressing scheme and routing maintenance, addressing children (TO)
- Reverse (PUSH) connection response (connecting out)
- HTTP/1.1 client and server for peer to peer transactions

Gnutella2 Standard for File Sharing

- All of the COMMON features listed in the previous section
- Operation in LEAF mode, additional node states OPTIONAL
- Some form of **bandwidth management** scheme to keep network and transfer bandwidth below 95% of the user's link capacity - be it manually configured or some automatic scheme (very important to avoid flooding local connection)
- **SHA1** and **TIGER ROOT URNs** for all shared objects
- XML metadata using existing schemas where appropriate (manual entry and peer acquired at minimum, automatic local collection highly recommended, service lookup optional)
- Universal 1-bit **query hash filter**, at least 2^20 length, intelligent density management scheme (superset combination required if supporting hub mode)

Gnutella2 Standard for File Sharing

- Gnutella2 object search mechanism, all client responsibilities and if supporting hub mode, server responsibilities too
- Local search processing including simple query language (Boolean operations, quoted search terms, numeric range searches, **interest flagging** (I), local rule-based metadata searching)
- Extensible hit format (URN/DN/MD/URL are REQUIRED, all other extensions OPTIONAL)
- HTTP/1.1 based upload system, URN based requesting, partial content requests, **active queuing**, **partial file uploading**, timestamp protected alternate source cache and exchange
- Tiger Tree volume calculation on shared files, caching on downloads, exchange via DIME. Local corruption detection OPTIONAL but recommended.

Network Components

- Node types and responsibilities for self-organizing network topology
- TCP stream connection handshake negotiation and compression encoding
- UDP reliable/semi-reliable transceiver stack and encapsulation protocol
- Gnutella2 common tree packet structure (basic protocol)
- Basic network maintenance packet types
- Known hub cache and **hub cluster** cache
- Node route cache and addressed packet forwarding
- **Query hash table**, superset table and exchange packet types
- Gnutella2 object search mechanism, client and server roles, forwarding rules, filtering rules, security
- Local search responder with simple query language and metadata
- HTTP/1.1 server for upload queuing and servicing
- HTTP/1.1 client for download scheduling and transfer
- **User profile challenge** and delivery packet types

Object Search in G2

- The search client selects an eligible hub from its **global hub cache** which has a recent timestamp, has not been contacted more recently than it allows, and has not yet been queried in this search.
- If a **query key** is not yet available for this hub, a query key request is dispatched.
- Once a query key is available, the search client sends a keyed query to the hub.
- Upon receiving the query, the hub checks the query key for **validity**.
- The hub then responds with a query acknowledgement packet, containing a list of **neighbouring hubs** which have now been searched and a list of **2-hop hubs** which have not yet been searched.
- The search client adds the list of searched hubs to its **"don't try again" list**, and adds the list of **"try hubs"** to the global hub cache for future selection.
- Meanwhile, the hub examines the query to make sure that it has not received it before. Duplicate queries are dropped.
- The hub then matches the query against the **query hash table** of all connected nodes, and sends it to those nodes which may be able to field a match.

Object Search in G2

- While this is occurring, the hub processes the query locally and returns any results it may have.
- Leaves directly attached to the hub which have a potential match will have received the query, and process it locally. They may elect to return results **directly to the search client**, or may return their results **to their hub for dispatch**.
- Other hubs in the **hub cluster** which received the query now examine it to ensure they have not processed it before. They do not send an acknowledgement.
- Assuming it is new, the hubs match the query against the query hash tables of their leaves but not their neighbouring hubs. Potential leaves receive a copy of the query, and the hub processes it locally.
- Once again, the hub returns its own results and may forward results for its leaves if they do not wish to dispatch them directly.
- Meanwhile, the search client receives any search results generated by the hub cluster.
- The search client makes a decision on whether it should continue its search. If so, the process starts again.
- **The search client will not requery any of the hubs in the hub cluster**, but it has a list of nearby hubs so that the crawl can continue.

References

- T. Klingberg and R. Manfredi, Gnutella 0.6, http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html
- A. Singla and C. Rohrs, Ultrapeers: Another Step Towards Gnutella Scalability Version 1.0, Lime Wire LLC, http://rfc-gnutella.sourceforge.net/src/Ultrapeers_1.0.html
- C. Rohrs, Query Routing for the Gnutella Network Version 1.0, Lime Wire LLC, http://www.limewire.com/developer/query_routing/keyword routing.htm
- BitTorrent, http://bittorrent.com/
- Magnet, http://www.magnetlink.org/
- Bitzi, http://bitzi.com/
- Gnutella2 Developers' Network (G2DN), http://www.gnutella2.com/