The Gnutella Protocol Specification v0.4[1]

Document Revision 1.2

Clip2
http://www.clip2.com
protocols@clip2.com

Gnutella[2] is a protocol for distributed search. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model. In this model, every client is a server, and vice versa. These so-called Gnutella *servents* perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. Due to its distributed nature, a network of servents that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline.

Protocol Definition

The Gnutella protocol defines the way in which servents communicate over the network. It consists of a set of descriptors used for communicating data between servents and a set of rules governing the inter-servent exchange of descriptors. Currently, the following descriptors are defined:

Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.
QueryHit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.

A Gnutella servent connects itself to the network by establishing a connection with another servent currently on the network. The acquisition of another servent's address is not part of the protocol definition and will not be described here (Host cache services are currently the predominant way of automating the acquisition of Gnutella servent addresses).

Once the address of another servent on the network is obtained, a TCP/IP connection to the servent is created, and the following Gnutella connection request string (ASCII encoded) may be sent:

GNUTELLA CONNECT/<protocol version string>\n\n

where <protocol version string> is defined to be the ASCII string "0.4" (or, equivalently, "\x30\x2e\x34") in this version of the specification.

---

[1] This document represents the de facto standard Gnutella 0.4 protocol. However, several implementations have extended the descriptors that comprise the protocol, and have imposed additional rules on the transmission of these descriptors through the Gnutella network. Known extensions to the protocol are provided in an Appendix at the end of this document, but some variations not documented here may be encountered in practice.

[2] Typically pronounced "new-tella" or, less commonly, "guh-new-tella".

A servent wishing to accept the connection request must respond with

GNUTELLA OK\n\n

Any other response indicates the servent's unwillingness to accept the connection. A servent may reject an incoming connection request for a variety of reasons - a servent's pool of incoming connection slots may be exhausted, or it may not support the same version of the protocol as the requesting servent, for example.

Once a servent has connected successfully to the network, it communicates with other servents by sending and receiving Gnutella protocol descriptors. Each descriptor is preceded by a Descriptor Header with the byte structure given below.

Note 1: All fields in the following structures are in little-endian byte order unless otherwise specified.

Note 2: All IP addresses in the following structures are in IPv4 format. For example, the IPv4 byte array

0xD0	0x11	0x32	0x04
byte 0	byte 1	byte 2	byte 3

represents the dotted address 208.17.50.4.

Descriptor Header

	Descriptor ID		Payload Descriptor	TTL	Hops	Payload Length	
Byte offset	0	15	16	17	18	19	22

Descriptor ID *A 16-byte string uniquely identifying the descriptor on the network*

Payload Descriptor
0x00 = Ping
0x01 = Pong
0x40 = Push
0x80 = Query
0x81 = QueryHit

TTL *Time To Live. The number of times the descriptor will be forwarded by Gnutella servents before it is removed from the network. Each servent will decrement the TTL before passing it on to another servent. When the TTL reaches 0, the descriptor will no longer be forwarded.*

Hops *The number of times the descriptor has been forwarded. As a descriptor is passed from servent to servent, the TTL and Hops fields of the header must satisfy the following condition:*

TTL(0) = TTL(i) + Hops(i)

Where TTL(*i*) and Hops(*i*) are the value of the TTL and Hops fields of the header at the descriptor's *i*-th hop, for *i* >= 0.

Payload Length *The length of the descriptor immediately following this header. The next descriptor header is located exactly Payload_Length bytes from the end of this header i.e. there are no gaps or pad bytes in the Gnutella data stream.*

The TTL is the only mechanism for expiring descriptors on the network. Servents should carefully scrutinize the TTL field of received descriptors and lower them as necessary. Abuse of the TTL field will lead to an unnecessary amount of network traffic and poor network performance.

The Payload Length field is the only reliable way for a servent to find the beginning of the next descriptor in the input stream. The Gnutella protocol does not provide an "eye-catcher" string or any other descriptor synchronization method. Therefore, servents should rigorously validate the Payload Length field for each descriptor received (at least for fixed-length descriptors). If a servent becomes out of synch with its input stream, it should drop the connection associated with the stream since the upstream servent is either generating, or forwarding, invalid descriptors.

Immediately following the descriptor header, is a payload consisting of one of the following descriptors:

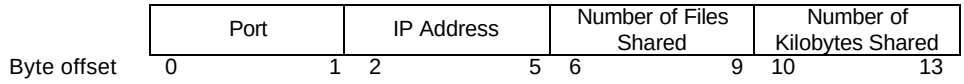
Ping (0x00)

Ping descriptors have no associated payload and are of zero length. A Ping is simply represented by a Descriptor Header whose Payload_Descriptor field is 0x00 and whose Payload_ Length field is 0x00000000.

A servent uses Ping descriptors to actively probe the network for other servents. A servent receiving a Ping descriptor may elect to respond with a Pong descriptor, which contains the address of an active Gnutella servent (possibly the one sending the Pong descriptor) and the amount of data it's sharing on the network.

This specification makes no recommendations as to the frequency at which a servent should send Ping descriptors, although servent implementers should make every attempt to minimize Ping traffic on the network.

Pong (0x01)



Port *The port number on which the responding host can accept incoming connections.*

IP Address *The IP address of the responding host.*

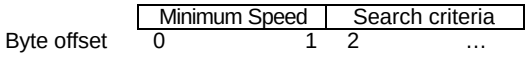
This field is in big-endian format.

Number of Files Shared *The number of files that the servent with the given IP address and port is sharing on the network.*

Number of Kilobytes Shared *The number of kilobytes of data that the servent with the given IP address and port is sharing on the network.*

Pong descriptors are only sent in response to an incoming Ping descriptor. It is valid for more than one Pong descriptor to be sent in response to a single Ping descriptor. This enables host caches to send cached servent address information in response to a Ping request.

Query (0x80)

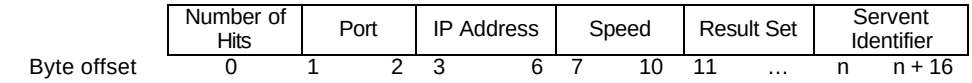


Minimum Speed *The minimum speed (in kb/second) of servents that should respond to this message. A servent receiving a Query descriptor with a Minimum Speed field of n kb/s should only respond with a QueryHit if it is able to communicate at a*

speed >= n kb/s

Search Criteria *A nul (i.e. 0x00) terminated search string. The maximum length of this string is bounded by the Payload_Length field of the descriptor header.*

QueryHit (0x81)



Number of Hits *The number of query hits in the result set (see below).*

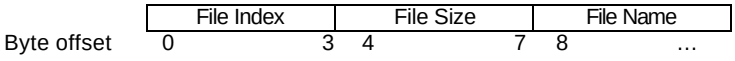
Port *The port number on which the responding host can accept incoming connections.*

IP Address *The IP address of the responding host.*

This field is in big-endian format.

Speed *The speed (in kb/second) of the responding host.*

Result Set *A set of responses to the corresponding Query. This set contains Number_of_Hits elements, each with the following structure:*



File Index *A number, assigned by the responding host, which is used to uniquely identify the file matching the corresponding query.*

File Size *The size (in bytes) of the file whose index is File_Index.*

File Name *The double-nul (i.e. 0x0000) terminated name of the file whose index is File_Index.*

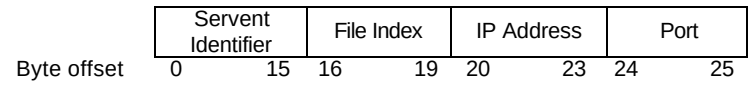
The size of the result set is bounded by the size of the Payload_Length field in the Descriptor Header.

Servent Identifier *A 16-byte string uniquely identifying the responding servent on the network. This is typically some function of the servent's network address. The Servent Identifier is instrumental in the operation of the Push Descriptor (see below).*

QueryHit descriptors are only sent in response to an incoming Query descriptor. A servent should only reply to a Query with a QueryHit if it contains data that strictly meets the Query Search Criteria.

The Descriptor_Id field in the Descriptor Header of the QueryHit should contain the same value as that of the associated Query descriptor. This allows a servent to identify the QueryHit descriptors associated with Query descriptors it generated.

Push (0x40)



Servent Identifier *The 16-byte string uniquely identifying the servent on the network who is being requested to push the file with index File_Index. The servent initiating the push request should set this field to the Servent_Identifier returned in the corresponding QueryHit descriptor. This allows the recipient of a push request to determine whether or not it is the target of that request.*

File Index *The index uniquely identifying the file to be pushed from the target servent. The servent initiating the push request should set this field to the value of one of the File_Index fields from the Result Set in the corresponding QueryHit descriptor.*

IP Address *The IP address of the host to which the file with File_Index should be pushed.*

This field is in big-endian format.

Port *The port to which the file with index File_Index should be pushed.*

A servent may send a Push descriptor if it receives a QueryHit descriptor from a servent that doesn't support incoming connections. This might occur when the servent sending the QueryHit descriptor is behind a firewall. When a servent receives a Push descriptor, it may act upon the push request if and only if the Servent_Identifier field contains the value of its servent identifier. The Descriptor_Id field in the Descriptor Header of the Push descriptor should not contain the same value as that of the associated QueryHit descriptor, but should contain a new value generated by the servent's Descriptor_Id generation algorithm. See the section below entitled "Firewalled Servents" for further details on the Push process.

Descriptor Routing

The peer-to-peer nature of the Gnutella network requires servents to route network traffic (queries, query replies, push requests, etc.) appropriately. A well-behaved Gnutella servent will route protocol descriptors according to the following rules:

1. Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor. This ensures that only those servents that routed the Ping descriptor will see the Pong descriptor in response. A servent that receives a Pong descriptor with Descriptor ID = *n*, but has not seen a Ping descriptor with Descriptor ID = *n* should remove the Pong descriptor from the network.

2. QueryHit descriptors may only be sent along the same path that carried the incoming Query descriptor. This ensures that only those servents that routed the Query descriptor will see the QueryHit descriptor in response. A servent that receives a QueryHit descriptor with Descriptor ID = *n*, but has not seen a Query descriptor with Descriptor ID = *n* should remove the QueryHit descriptor from the network.

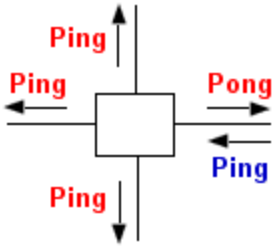
3. Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptor. This ensures that only those servents that routed the QueryHit descriptor will see the Push descriptor. ~~A servent that receives a Push descriptor with Descriptor ID = *n*, but has not seen a QueryHit descriptor with Descriptor ID = *n* should remove the Push descriptor from the network.~~ A servent that receives a Push descriptor with Servent_Identifier = *n*, but has not seen a QueryHit descriptor with Servent Identifier = *n* should remove the Push descriptor from the network. Push descriptors are routed by Servent_Identifier, not by Descriptor_Id.

4. A servent will forward incoming Ping and Query descriptors to all of its directly connected servents, except the one that delivered the incoming Ping or Query.

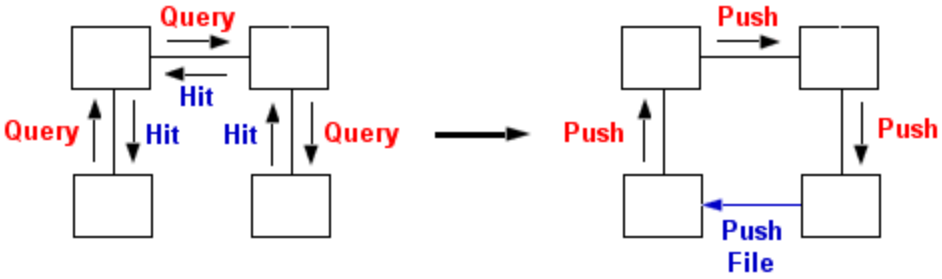
5. A servent will decrement a descriptor header's TTL field, and increment its Hops field, before it forwards the descriptor to any directly connected servent. If, after

decrementing the header's TTL field, the TTL field is found to be zero, the descriptor is not forwarded along any connection.

6. A servent receiving a descriptor with the same Payload Descriptor and Descriptor ID as one it has received before, should attempt to avoid forwarding the descriptor to any connected servent. Its intended recipients have already received such a descriptor, and sending it again merely wastes network bandwidth.



Example 1. Ping/Pong Routing



Example 2. Query/QueryHit/Push Routing

File Downloads

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. Files are downloaded out-of-network i.e. a direct connection between the source and target servent is established in order to perform the data transfer. File data is never transferred over the Gnutella network.

The file download protocol is HTTP. The servent initiating the download sends a request string of the following form to the target server:

GET /get/<File Index>/<File Name>/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella\r\n[3]
\r\n

where <File Index> and <File Name> are one of the File Index/File Name pairs from a QueryHit descriptor's Result Set. For example, if the Result Set from a QueryHit descriptor contained the entry

File Index	*2468*
File Size	*4356789*
File Name	*Foobar.mp3\x00\x00*

then a download request for the file described by this entry would be initiated as follows:

GET /get/2468/Foobar.mp3/ HTTP/1.0\r\n

Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella
\r\n

The server receiving this download request responds with HTTP 1.0 compliant headers such as

HTTP 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 4356789\r\n
\r\n

The file data then follows and should be read up to, and including, the number of bytes specified in the Content-length provided in the server's HTTP response.

The Gnutella protocol provides support for the HTTP Range parameter, so that interrupted downloads may be resumed at the point at which they terminated.

Firewalled Servents

It is not always possible to establish a direct connection to a Gnutella servent in an attempt to initiate a file download. The servent may, for example, be behind a firewall that does not permit incoming connections to its Gnutella port. 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 descriptor describing the target file. The servent that is the target of the Push request (identified by the *Servent Identifier* field of the Push descriptor) should, upon receipt of the Push descriptor, attempt to establish a new TCP/IP connection to the requesting servent (identified by the *IP Address* and *Port* fields of the Push descriptor). If this direct connection cannot be established, then it is likely that the servent that issued the Push request is itself behind a firewall. In this case, file transfer cannot take place.

If a direct connection can be established from the firewalled servent to the servent that initiated the Push request, the firewalled servent should immediately send the following:

GIV <File Index>:<Servent Identifier>/<File Name>\n\n

Where <File Index> and <Servent Identifier> are the values of the File Index and Servent Identifier fields respectively from the Push request received, and <File Name> is the name of the file in the local file table whose file index number is <File Index>. The servent receiving the GIV request header (i.e. the Push requester) should extract the <File Index> and <File Name> fields from the header and construct an HTTP GET request of the following form:

GET /get/<File Index>/<File Name>/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella\r\n[3]
\r\n

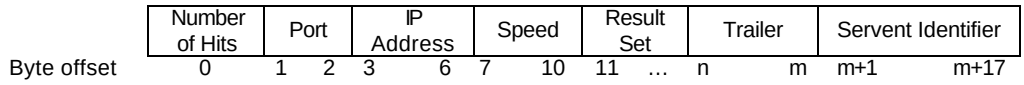
The remainder of the file download process is identical to that described in the section entitled "File Downloads" above.

[3] The allowable values of the User-Agent string are defined by the HTTP standard. Servent developers cannot make any assumptions about the value here. The use of 'Gnutella' is for illustration purposes only.

Appendix 1: Gnutella Protocol Extensions

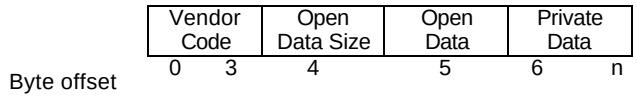
Extended Query Hit Descriptor (Description Updated 03/15/2001)
First introduced by BearShare v1.3.0, the extended QueryHit Descriptor extends the original Gnutella QueryHit descriptor by placing extra data between the last double-nul terminated filename of the Result Set and the Servent Identifier. An extended QueryHit descriptor will have the following payload structure:

QueryHit (0x81)



Where the BearShareTrailer field has the following structure:

Trailer



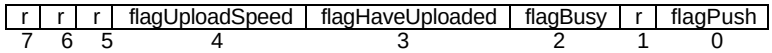
Vendor Code Four case-insensitive characters representing a vendor code. Recognized vendor codes are as follows:

Vendor Code	Application Name[4]
BEAR	BearShare
LIME	LimeWire
TOAD	ToadNode
GNOT	Gnotella
MACT	Mactella
GNUC	Gnucleus
GNUT	Gnut
GTKG	Gtk-Gnutella
NAPS	NapShare
OCFG	OpenCola
HSLG	Hagelslag
CULT	Cultiv8r

Open Data Size Contains the length (in bytes) of the Open Data field.

Open Data Contains two 1-byte flags fields with the following layout and in the specified order:

flags



flagUploadSpeed = 1 if and only if the *flagUploadSpeed* flag in the *flags2* field is meaningful.

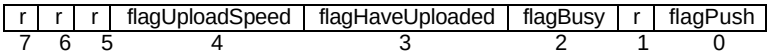
flagHaveUploaded = 1 if and only if the *flagUploaded* flag in the *flags2* field is meaningful.

flagBusy = 1 if and only if the *flagBusy* flag in the *flags2* field is meaningful.

flagPush = 1 if and only if the servent is firewalled or has not yet accepted an incoming connection.

r = reserved for future use.

flags2



flagUploadSpeed = 1 if and only if the Speed field of the QueryHit descriptor contains the highest average transfer rate (in kbps) of the last 10 uploads.

flagHaveUploaded = 1 if and only if the servent has successfully uploaded at least one file.

flagBusy = 1 if and only if the all of the servent's upload slots are currently full.

flagPush = 1 if and only if the *flagPush* flag in the *flags* field is meaningful.

r = reserved for future use.

[4] More information on these applications can be found at http://www.gnutelliums.com.

Private Data	Undocumented BearShare-specific data. The length of this field can be determined as follows: QueryHit Descriptor Payload Size - (OpenDataSize + 4 + 1).

One way for developers to handle the extension is to

(1) Be aware that an incoming QueryHit may or may not contain additional data after the result set and before the Servent Identifier. No complete specification exists for the number of bytes that may be present, or their content. Use the Payload Length field and count bytes as they are read from the stream to determine whether the extension bytes are present.

(2) If they are, read them from the stream, leaving 16 bytes for the Servent Identifier.

(3) Process the QueryHit as usual.

Gnotella

Versions of the Gnotella client at least as early as 0.73 (released July 30, 2000) place extra data in QueryHit descriptors. According to the Gnutella 0.4 protocol specification, each element of the result set in a QueryHit descriptor is terminated by a double-nul. Gnotella may place extra data between the two nuls. Although its exact layout is unknown, this data represents the bit-rate, sample rate, and playing time of the MP3 file described by the result set entry. If the file described by the result set entry is not an MP3 file, there is no data placed between the nuls.

Some servents may process Gnotella's extended QueryHit descriptors without adverse consequences by simply disregarding data between nuls. Others may, upon not finding an element of the result terminated as expected, improperly read the descriptor. Once misread, it may be subsequently mishandled, and the connection that delivered it may be disconnected.