Optimization of Gnutella Protocol
Subhransu Dash, Radha Tomar

ABSTRACT
The technology of peer to peer has improved the download mechanism in high rate. The total number of nodes keeps increasing in the network with improvement in the peer to peer protocol. In this discussion we observe the operation of Gnutella protocol and the optimization of its descriptor ping pong message passing mechanism. This also focuses on the scalability of the mechanism. Security issues are also considered with few statistics reflecting on the future changes.

KeyWords
Peer to Peer network; Gnutella protocol; ping pong mechanism;
INTRODUCTION

Gnutella I is a peer to peer protocol. This is an open protocol, this protocol mainly used for file searching and sharing, decentralized in group members. The search protocol addresses the sharing and searching of group membership of files. This term represent entire group of the computers which will have Gnutella application installed in the virtual network.

Peer-to-peer networks are primarily based on the resources and characteristics of the client nodes that compose them rather than those of a (or set of) server(s). They use ad-hoc connections to link these nodes. In an exceedingly pure peer-to-peer network (decentralized) there is no separation between shoppers and servers and every functionality is accomplished by peers. In the contrary, in an exceedingly client-server model (e.g. FTP

Server) all communication and data transfer is bypassed through the central server.

There is a sort of networks (e.g. Napster, OpenNAP) whose operations are part relied on a client-server structure (e.g. searching) and part on a peer-to-peer structure (e.g. file transfer). But, for example, Gnutella and Freenet swear utterly on peer-to-peer structures [7]. Peer-to-Peer networks are mainly used for sharing resources, like files ([3], [4]) and computing cycles ([5]), or as a platform for cooperative tasks. The central functions perform by a peer are:

• Share and seek for resources
• Use a particular protocol to implement the resource sharing: establish and maintain connections with peers settle for user input and initiate protocol messages, method and route protocol messages [6]

GNUTELLA V0.4

In a Gnutella network, every client is server and every server is a client, too. Thus, each node is known as a servant and accomplishes each type of needed operations. It can perform queries and invoke the results, as long as respond to queries comparison them to its own repository. This way, the centralized networks’ main problem of single-point failure on the central server-broker or indexer node(s), is bypassed. The servants communicate with the subsequent 5 descriptors (types of messages):

• Ping, for discovering other host servants
• Pong, for responding to Pings with the address and also the data sharing info of the sender servant
• Query, for looking shared info among other servants
• QueryHit, for responding to Queries upon a sure-fire match
• Push, for permitting firewalled servants to share

They are inserted to the network in the beginning, victimization host cache services to find the primary host and so establish with it a TCP/IP affiliation. Later they sent a connection request string, which, just in case of its acceptance (if e.g. there are accessible slots), is responded with a affiliation acceptance string.
The descriptors’ messages begin with a header, that contains the Descriptor ID (a unique 16-bit string), the Payload Descriptor (for 1 of the 5 on top of mentioned types), the TTL (Time To Live—number of remaining times it may be forwarded to alternative servants), the Hops (the present variety of times it has been forwarded) and therefore the Payload Length (length of the data following the header). The right obtain of the initial TTL will play important role on the matter of managing the traffic on the network, as it is that the solely manner the messages may be terminated. Succeeding descriptor to be read is found Payload Length bytes after the tip of the present header, thus simplifying the whole method of synchronizing and sustentative the input stream of messages on the servant. [2]

A Ping has no payload and 0 lengths. A pong includes the Port for incoming connections, the information science address and also the number and total size of shared files of the responder. A query transmits the required minimum transfer speed for responding servants and also the search criteria string. A QueryHit carries the number of hits on the specific node, the Port for incoming connections, the information science address, the transfer speed and the identifier of the servant, and also the result set. This set contains the resulted files listed with their File Index (unique identifier within the responder’s shared files index), their size and their name.

The Descriptor ID’s of associated Queries and QueryHits should be a similar. The Pongs travel back on a similar route that the corresponding Pings came from (see Fig.A) and the same happens for Queries and QueryHits (see Fig.B). Pings and Queries are forwarded to any or all direct neighbor nodes, except the sender. Upon receiving a message, a servant decreases the TTL and will increase the Hops by one, so when TTL becomes 0, the descriptor seizes to be retransmitted. a similar descriptor is forwarded only once from a similar servant, undeniable fact
that results in lowering the overall message traffic and thus to a system performance benefit. The files from the result set are downloaded from the requester with an immediate association to the source mistreatment http. Interrupted downloads is resumed from where they stopped [3].

**Fig (B): Query Propagation [9]**

**Web Service Discovery with DAML-S on Gnutella**

A Web Service linguistics description language (DAML-S/OWL-S) and a redistributed P2P network (Gnutella), plus the linguistics matching rule, are all together accustomed implement the example conferred in paper [8]. There, they designed the mix of the DAML-S matching with the gnutella messages. Consistent with that, a node sends a Gnutella question to identify a service provider, and this request is forwarded between neighboring peers as a file request would. once the request is matched to a service provided by a receiver peer, a QueryHit massage is send back to the requester, containing the address of the W.S. every peer ought to thus give DAML-S descriptions of its hosted services, a DAML ontology program (for performing the matching) and a Gnutella discovery application.

Specifically, as we see in fig below, the first layer is that the DAML-S computer programme, whose input is DAML ontologies and DAML-S descriptions (DAML files) and output a collection of arguments. This set is that the input for the DAML-S processor of the second layer, which separates the terms in keeping with their kind in: Profile and Matchmaking Rules to be used in the discovery process, and process
Model and Grounding Rules to be used in the service invocation. It also communicates with the front application which provides the input for the querying and therefore the interaction with different services. The third layer declares the discovery and invocation ports for the online Services, severally with the second layer separation.

The application initiates an internet Service request which is formalized and transmitted through the processor. The receivers compare the question to their service description repository using their Matchmaking rules and answer with successful message. The hit(s) are returned to the appliance which chooses to invoke (one of) them, following the provider’s Process Model and Grounding rules.

**DRAW BACK OF GNUTELLA**

There are number of problems with this protocol

- There are not any rules for generating a globally distinctive message identifier
- The network breaks if constant ids show up. Why not use information processing and port number of the servent as base for the identification. Most servent implementations do, but again, there are not any set rules to get a message id.
- The servent has to keep a history of everything. This is very memory overwhelming. Of course, a servent might forget previous messages, but no one knows however what previous is, and it still has to keep a history.
- The routing of 'push' messages is obvious. Why use a 16 byte servent identifier, after you have a four byte IPv4 address and a pair of byte port number?

The protocol is very compact (but inconsistent in few areas), everything is encoded during a few bytes. but in the finish, however, we tend to resort from the HTTP ASCII protocol. It is assumed that initial developer had some HTTP transfer code pre available around or something.
On the other hand, HTTP is good because currently it will connect to a servant employing an applications programm. In applications, it fails, because most servents just have all their links purpose to a website wherever it is able to transfer their servant binary (which is not therefore smooth).

Also, if the servant rescans his shared directories just before somebody starts a transfer, he won't be ready to get the file because the file number has changed. This does not happen a lot, but it will happen in typical. If a servant disconnects and re-connects, it gets a current servant id and it has lost its history. This has its consequences for users that were progressing to transfer similar files.

Note that in an exceedingly topology unaware network, there is no would like for a ping and pong message. It has been enforced as a result of people need to grasp what number of servents are out there and the way many bytes of information are out there.

Technically speaking, the ping and pong are of no use and solely wasting bandwidth. The ping message generates thousands and thousands of pong messages, and since a PING is routed to everybody, one PING message effectively makes everyone ping everyone. not to mention the fact that some servant implementations do periodic pinging.

To help out users with a slow dial-up connection, the Gnutella backbone was unreal. The backbone consists of many hosts acting as host cache. In practice, everybody simply connects to a number cache a let it do the task. As a result, the Gnutella network is quickly turning into a little network with many giant servers rather than that includes a chaotic structure.

PROPOSED SOLUTION
Loose the PING message. It’s tiny so it is in all probability be not wasting abundant bandwidth, however for sure there are better ways that to try to do this. Since a servant is aware of if it already has seen a message or not, we will let a message with a grand total go round and let everybody update the grand total. This fashion we tend to automatically have periodic updates on what the full size of the network is, to stay net from flooding adore it does with PING, this message should be initiated as long as (and solely if) the servant hasn't seen a message describing a complete for the later 5 minutes. When the message is supposed to reach its TTL or when a servant has no one else to forward the message to, a replacement message of a special sort should be broadcasted backwards so that everybody gets the most recent updated value of the grand total. This value is not updated anymore, when a short time it dies out and when the timeout is reached somebody can transfer a replacement ping message to demand a recount of the grand total.

This works best if the servents wait to collect totals from variety of ports, instead of straightaway forwarding the utmost to everybody. This is different from the present PING and malodor implementation, with PING you broadcast to everybody and find a flood of routed replies back. With the technique represented here, the reply is also broadcasted in order that everybody can fancy the updated grand total. It’s true that you might receive all types of different totals, you should pick the highest worth, and lower values should be born. It still costs a lot of information measure, but at least you're not routing all PONGs back to everyone (since everyone pings). Instead, you're taking part in a continuous world broadcast.

CONCLUSION
The Gnutella network is extremely nice, but things will be done higher. It is arduous to change an existing protocol, especially once you already have thousands of users. However, exploitation the name 'Gnutella', it should be possible to urge the globe to go use the newest improved version of this network.

REFERENCES


