Topology of Gnutella Network: Discovery and Analysis

Yugo Nakai, Matei Ripeanu, Xuehai Zhang

Abstract

To be written

1. Introduction

1.1 Peer-to-peer definition

Peer-to-peer networks (known simply as "P2P") allow individual computer users to communicate directly with each other, sharing information and resources. Webopedia [3] defines P2P architecture as:

> "A type of network in which each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures, in which some computers are dedicated to serving the others […]"

Recent P2P applications try to provide the infrastructure to build communities that share computer cycles (Seti@Home, Entropia) and/or storage space (Napster, FreeNet, Gnutella). On top of these, collaborative environments (Groove[1]) can be created. Two factors have fostered the recent explosive growth of P2P applications: first, the low cost and availability of huge computing and storage resources, and second, increased network connectivity.

Early P2P applications like Napster (the first popular P2P file sharing system) do not fully comply with the P2P definition. They still rely on centralized databases to index files and respond to searches; only file transfers are directly between peers. Newer applications (Gnutella, FreeNet) allow users to search for and swap files without recourse to traditional databases and servers. They are *completely decentralized:* each member of the network performs the same set of tasks and "serves" other members.

A common characteristic of this new breed of applications is that they build a virtual network at the application level with its own routing and transport mechanisms. The characteristics of this virtual network determine the most important proprieties of the P2P application/service, such as application performance and reliability as well as other requirements like anonymity.

1.2 Gnutella

In [1] Eytan Adar and Bernardo Huberman analyze network content and traffic and tries to infer user behavior in the Gnutella network. For this report, we chose to analyze Gnutella's virtual network topology and try to investigate possible improvements to the protocol that would allow better scaling and increased reliability.

Gnutella attempts to use the P2P paradigm to fulfill the following design goals:

o Performance (aggregate storage size, response time, availability) – Ideally, storage space and file availability increases linearly with the number of nodes, while user interest improves exponentially after a threshold value. The non-linearity of the topology should allow searches to improve speed and probability of success linearly with nodes and exponentially with graph diameter.

o Scalability – Interestingly, Gnutella was not intended by its initial creator to scale beyond a few hundred or thousand nodes, but to a large extent, it has successfully survived into the thousands and above. This is discussed later in this paper.

o Anonymity (for contributors, for readers) (FreeHaven, FreeNet). Anonymity is valued by Gnutella as a means to protect privacy of people seeking or providing information that may not be popular (or even legal).

o Reliability (FreeHaven) – Especially in an anonymous environment, robustness is important. Loss and/or corruption of data and maintenance packets should not cause severe problems.

o Ability to live in a dynamic environment – Again, anonymous environments tend to have less incentive for people to maintain a "presence," unlike the Web for instance, so nodes are much more likely to join or leave the network in a short time. There is also the potential for censorship to shut down individual nodes. So, Gnutella aims for flexibility to keep operating transparently despite a constantly changing network.

In a Gnutella network, each node (*servant*[1] in Gnutella terminology) maintains a list of neighbors, thus creating a virtual network of *servants* at application level. All communication is done between adjacent servants. One could view this as a (undirected) graph definition stored in a distributed manner: each node (*servant*) stores its adjacency (neighbor) list.

1.3 Our approach

We collected the graph definition using a crawler, performed structural graph analysis, investigate the dynamic graph structure over time and, finally, used these results to come up with possible improvements of the Gnutella protocol.

Short summary: in Section 2 we present a brief description of the protocol. Section 3 describes the tools we have developed and used to gather data. Section 4 is a collection of question we have tried to answer. In Section 5 we briefly summarize out findings and comment on Gnutella's weak/strong points while Section 6 proposes protocol improvements. We conclude in Section 7.

2.   Gnutella's protocol description

The 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. Servents are also responsible for managing the background traffic that spreads the information used to maintain network integrity

---

[1] In Gnutella terminology a servant is a service with server *and* client capabilities

Since there are no central servers in the Gnutella network, in order to join the system a new servent initially connects to one of several known hosts that are almost always available [2] (although these generally do not provide shared files). These hosts then forward the IP and port address information to other Gnutella peers.

Once attached to the network (having one or more open connections with servents already in the network), peers interact with each other by means of messages. Peers will create and initiate a broadcast of messages as well as re-broadcasting others (receiving and transmitting to neighbors). In case of a broadcast the *servent* will send the message to all peers it has open connections with (of course excepting the peer that originated the message).

 The messages allowed in the network are:

> *Ping* Messages - Essentially, an "I am X, who's out there?" message that will be broadcasted (with a limited time to live) into the network. Peers forward this kind of message to their neighbors so that it is possible to later find other peers. This is needed in case the node gets disconnected from the network.

> *Pong* Messages - A reply to a ping ("I am here!"). The pong message contains information about the peer such as its IP address and port as well as the number of files shared and the total size of those files. These messages are propagated backwards along the path that the PING message originally took.

> *Query* Messages - These are messages stating, "I am looking for file x" and are also broadcasted in the network. Query messages are uniquely identified, but their source is unknown.

> *Query Response* Messages - These are replies to query messages, and they include the information necessary to download the file (IP, port, and other location information). Responses also contain a unique client ID associated with the replying peer. These messages are propagated backwards along the path that the query message originally took. Since these messages are not broadcast it becomes impossible to trace all query responses in the system.

> *Get/Push* Messages - Get messages are simply a request for a file returned by a query. The requesting peer connects to the serving peer directly and requests the file. Certain hosts, usually located behind a firewall, are unable to directly respond to requests for files. For this reason the Gnutella protocol includes push messages. Push messages request the serving client to initiate the connection to the requesting peer and upload the file. However, if both peers are located behind a firewall a connection between the two will be impossible.

Several features of Gnutella's protocol prevent messages from being re-broadcast indefinitely through the network. First each message has a randomly generated identifier. Since the identifier size is 16 bits and messages are short lived the system considers them

---

[2] One such server is www.gnutellahosts.com. In fact each user community has to have at least one of these servers. Japanese Gnutella user community have their server at www.jnutella.org:8000

as uniquely identifying a message. Second the *servent* keeps a short memory of messages that have been routed thus preventing re-broadcasting. Additionally, messages are flagged with a time-to-live (TTL) field. At each hop (re-broadcast) the TTL is decremented. As soon as a peer sees a message with a TTL of zero, the message is dropped (i.e. it is not re-broadcast). In addition, a "HOPS" integer field is incremented on each hop. Peers have the right to drop (ignore) any message that has a TTL or HOPS above a user-specified limit, thus preventing excessive traffic generated by a single query.

To summarize: to become a member of the network a servent has to open one (or many) connection with a *servent*(s) that is already in the network. One design requirement for Gnutella was to operate in an extremely dynamic environment: nodes decide often to join or to leave[3], network connections are unreliable. To cope with requirements, after joiing the network network a *servent* will periodically 'ping' its neighbors to find about other participating *servents*. This information allows connecting to other *servents* (and thus reconnecting to the network) in case all *servents* he is connected thru decide to leave. We will provide in section 4 an estimate of this overhead traffic necessary to preserve network consistency.

3.   Tools

We developed a network crawler to discover Gnutella's network topology and we created modules for network analysis. To have a better image on the topology we used ATT's - open source graph drawing software Graphviz.

3.1   Gnutella network crawler

Starting points of access to Gnutella network are publicly available at: http://www.gnutellahosts.com/. The crawler starts by connecting to one of these nodes (pairs host:port) and sends a PING message with TTL=2. It receives back PONG from the node he is connected to and from nodes connected to this one. This way the crawler will know (1) all graph edges around the node it contacted, (2) a list of new nodes to investigate. The crawler will recursively discover the graph and contact all its nodes until no new nodes are found. Of course we collect all additional information available: graph nodes are labeled with the number of files they share and their size and we could label links with their declared capacity.

The following is the pseudocode for the crawler:

```
Procedure GraphCrawl (StartPointsList)
    graph  :=  Null
    nodesToDiscover = StartPointsList
    while (not empty(nodesToDiscover)):
        node := Head(nodesToDiscover)
        nodesToDiscover := Tail(NodesToDiscover)
        conn := EstablishConnection(node)
        SendPing(conn, TTL := 2)
        for  timeout seconds do:
```

---

[3] A survey [5] published by dss.clip2.com provide some insight: they report that 50% of the nodes leave the network in less than five hours while 25% of the nodes remain in the network more than one day.

```
            adjacencyList := Null
            msg := ReceiveMessage(conn)
            if (MsgType(msg) = PONG) then
                  newNode = MsgHost(msg)
                  adjacencyList.append(newNode)
                  if  (not (AlreadyDiscovered(newNode)) then
                        NodesToDiscove.append(newNode)
                  Endif
            endif
        Graph.append((Node, adjacencyList))
    end while
    return graph
  end procedure
```

We developed first a sequential version of the crawler. With this we empirically discovered optimal values for connection establishment timeout as well as for connection listening timeout (the time interval the crawler will listen to receive PONGs after it sent a PING). With successive crawls we assembled a starting points list with about 10,000 entries.

However, allowing 30s to wait for PONG messages and 10s connection establishment timeout leads to crawling times of 20 to 30 hours for a medium sized Gnutella network. To make things worse the dynamic behavior of nodes makes sure that result of our crawl is far from a snapshot of network topology. With this goal in mind we developed a **distributed crawling strategy**. The distributed crawler has a client server architecture: the server is responsible with assembling the final graph and giving work to clients wile the clients receive a small list of starting points and discover the network topology around these points. Although we could potentially use a large number of clients (easily in the order of hundreds) we decided to use only 10 clients to reduce the invasiveness of our search.

3.2   Graph analysis

Analyzing Gnutella's network topology is important to help us understand the connectivity or coordination of the nodes. We hope we will be able to come up with protocol improvements that will generate optimized network topologies.

We focus our analysis first on general graphs proprieties (like number of nodes, number of edges, graph connectivity, etc.) second on Gnutella's particular aspects (usage patterns, generated traffic).

The general issues covered in our analysis are:

- **Fake nodes distribution:** figure out those nodes with false IP addresses. Provide the information about the neighbor nodes of each fake node.

- **Graph Connectivity:** find all the connected components in a certain graph. Figure out their size, member nodes.

  Get statistics on connectivity degree (average, maximum connectivity) distribution. Figure out the share of each degree and its member nodes.

- **Information distribution**: Get statistics on information distribution. Figure out what part of the nodes support certain range of shared file number or size.

- **Domain distribution** Get statistics on domain distribution. Figure out the share of each domain (including those TDNS such as COM,EDU and those country domains) and its member nodes.

- **Usage patterns**

- **Generated traffic analysis**

We develop the analysis tool to take the graph generated by the graph crawler as the input. We first construct the new graph formatted upon an adjacent table data structure. Since the original graph contains nodes fake (for example servents that declare port 0 as the port they are listening) we discarded instead of inserting them to the nodes of the new graph. The implementation of our analysis tool involves a lot of graph-theoretical techniques, such as depth first search, shortest path algorithm, etc.

3.3 Graph drawing

In order to visualize the structure of the Gnutella network, we employed AT&T Labs - Research's GraphViz toolset. We created scripts to translate our graph data into a format readable by DOT and NEATO, the primary GraphViz tools. This allowed us to produce two kinds of graphs: hierarchical and space-optimized (please refer to Appendix 1 for or follow the hyperlinks samples).

The hierarchical view more clearly shows "nexus" nodes of high connectivity and "single" nodes with degree = 1 or 2. The space-optimized view presents a more symmetrical view of the network, with inner nodes generally corresponding to nexuses and edge nodes corresponding to single nodes; network diameter can be roughly visualized by inspection. Nodes are labeled with numbers that (in the source file) are mapped to IP addresses, allowing us to visually locate nodes of interest and look up its data.

The hierarchical view retains more clarity as the graph size increases, compared to the globular space-optimized view, but both quickly become unreadable beyond approximately 300 nodes (the size of the two examples). Nonetheless, the smaller graphs allow us to observe patterns and intuitively grasp the structure as a whole; we can also attempt to generalize our observations from smaller graphs to the less readable large graphs.

4. Network topology and analysis

The protocol specifies that a node that wants to join the network has to contact one of the nodes already in the network. It does not specify how this start node is found. The common procedure to obtain a start node is either to search through web pages that provide this information or to connect to special services like gnutellahosts.com that. In both cases the network tends to cluster around the nodes provided as start points. In the

following we investigate thoroughly what appears to be the most active component: the component centered on gnutellahosts.com start service.

4.1   Maps

See 3.3

4.2   How big is the network?

We ran the crawler to generate graph data during a one-month period. As you may see from the following figure there is a steep increase in network size after November 20[th.] We believe this is not due to a change in Gnutella's network but we link it to an improved crawler version (we built a distributed crawler that is much faster).
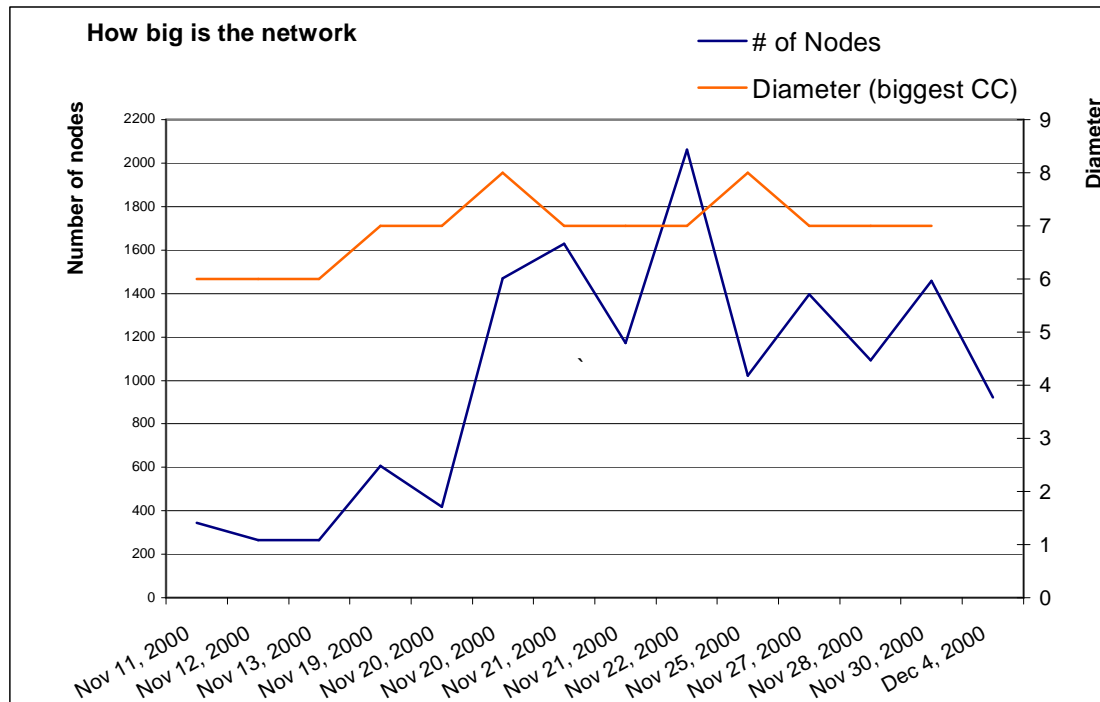

Figure 3: Graph size and diameter (data collected over a three week period, 14 measurements)

Knowing the graph diameter allows tuning the probability of false negatives in finding a document by tuning the time to live (TTL) of a query.

We see that the diameter, which is the longest short distance between two nodes, always fells to smaller values, such as 7 or 8. The reason for this might be that Gnutella servants typically issue ping messages with TTL= 7.

4.3   How much information is out there? How is the information distributed?

Gnutella is a file sharing application. Therefore evaluating the amount of information the application   manages is crucial in understanding it. Next figure plots the amount of information shared in the networks we investigated over the same two week period:
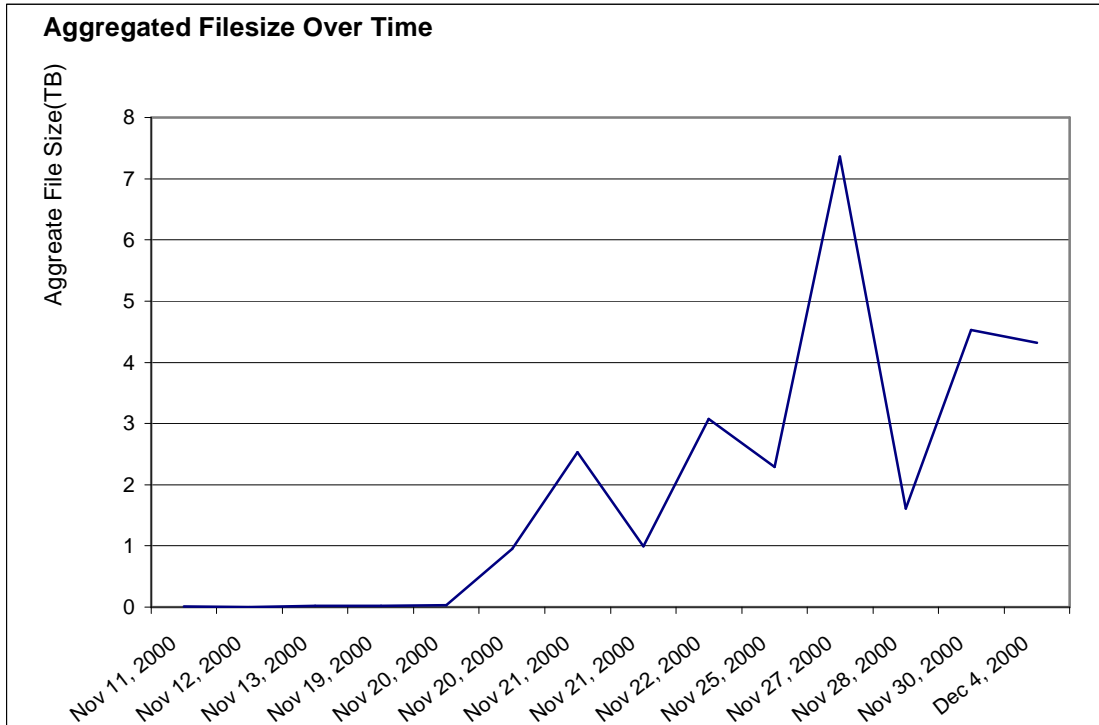
Aggregated Filesize Over Time

Figure 4: Aggregated size of data shared (data collected over a three week period, 14 measurements)

The aggregated size of shared information typically goes up to several TB. But most information is stored among a small percent of the nodes and a large percent of the nodes have no information to share. We validate results presented by [1] that show that most users are only information consumers. If a larger percent of the users shares files we would expect a different network topology: a more balanced one (as you will se in section 4.7 a large number of nodes have degree 1 while there is a small number of hosts that have a large number of connections).
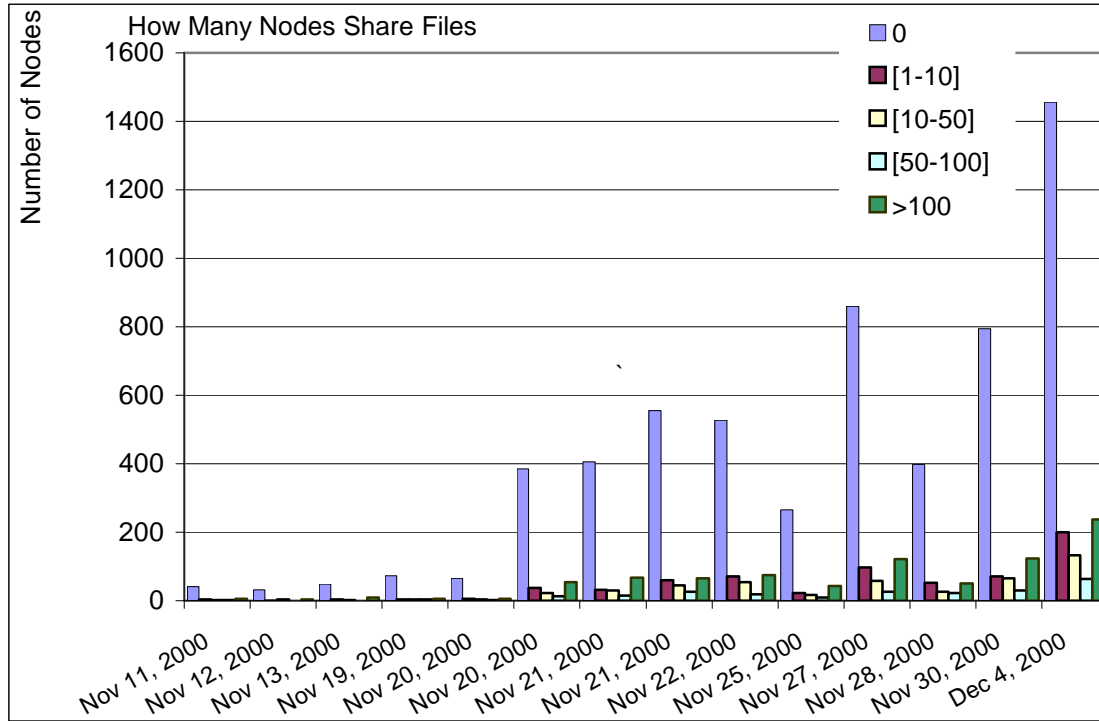
Figure 5: Number of files shared by each node (data collected over a three week period, 14 measurements)

4.4   How dynamic is the network

The protocol is designed to handle a highly dynamic environment. We tried to obtain a measure of how the network topology is given this environment:

- *Node lifetime*: [6] reports (and our measurements confirm) that 40% of the nodes quit the network in less than 4 hours. They also find that there are some 'persistent' nodes (about 20% of the total) that tend to be present in the network for more than 24 hours.

- *Link lifetime*: By connecting repeatedly to the same node we observed that links (TCP connections between two servents) have a very short lifetime. Some empirical measurements show that only about 15% of the links survive more than one hour and les than 3% survive more than one day.

As an example, we created an animated .GIF of one node and its immediate neighbors over a span of 157 minutes, by checking the node approximately every 10 minutes. (The chosen node is marked as a blue double circle.) It is clear that even in these short intervals, the graph topology can change rapidly. This suggests that even if graph diameter is fairly large compared to TTL, a few repeated searches at short intervals should be able to reach most nodes in the graph. If true, this would allow us to achieve effectively larger file availability without the high overhead of a small-diameter graph. (Of course, this is counterbalanced by the increased possibility that a positive search

result will be unable to return to the initiating servant, since results are returned by back-propagation.)

4.5 Connectivity analysis

Connectivity (connected components, degree of the nodes, etc) takes an essential role in the topology of Gnutella network because one user can search file only within the network component she is member of. Next figure shows the number of connected components our crawler found. We have to mention that usually we find a large component (containing about 90% of the nodes) plus some other very small components.
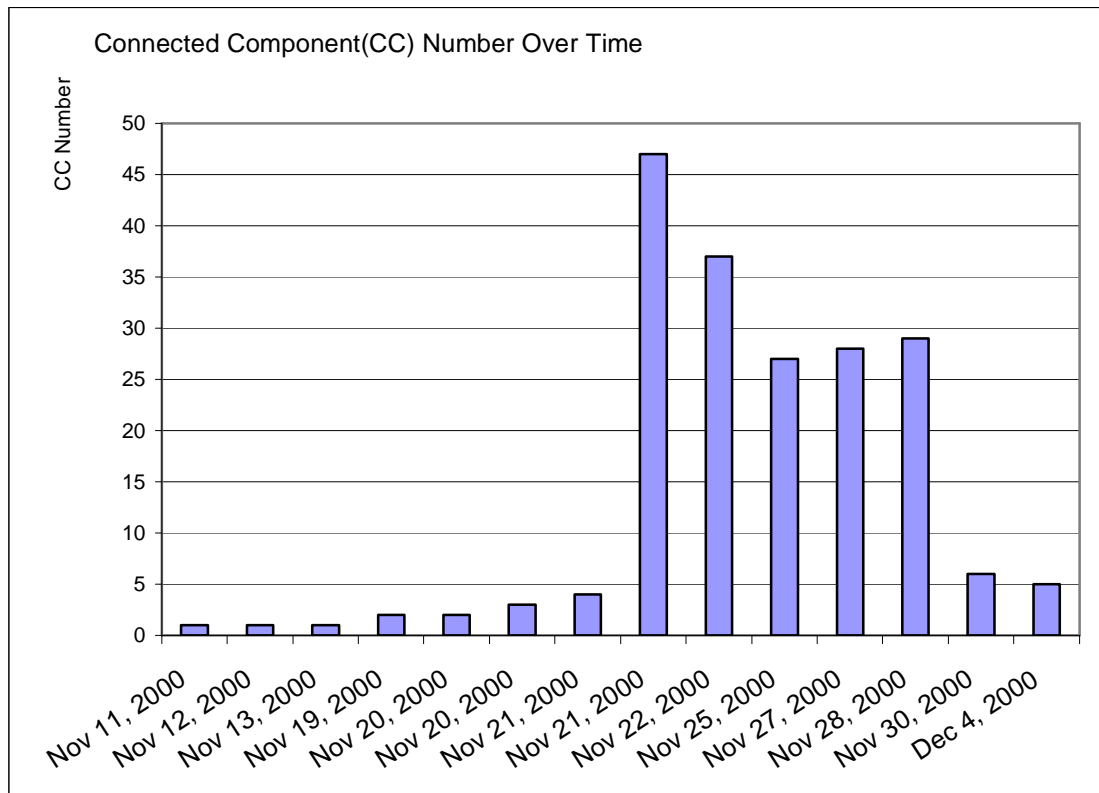

Figure 7: Connected component number over time (data collected over a three week period, 14 measurements)

In figure 8 we show average and maximum node connectivity for the topologies we discovered:
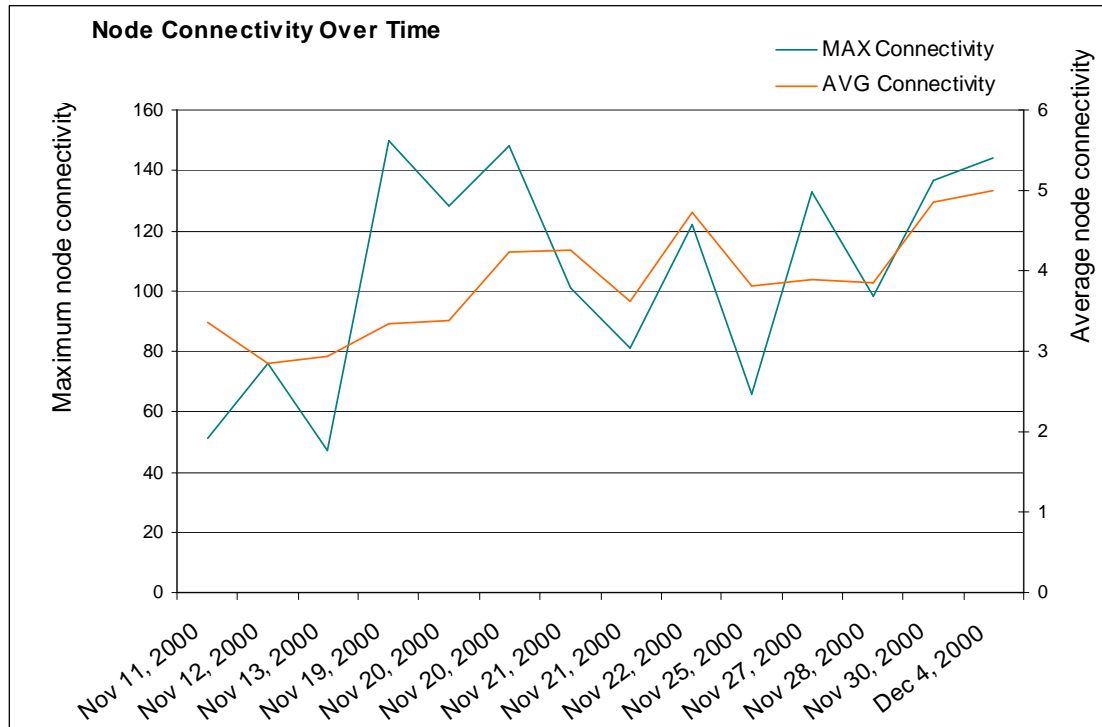
Figure 8: Average and maximum node connectivity (in the biggest connected component)

To emphasize the fact that the number of connections a node has is unevenly distributed we show the distribution of nodes by connectivity. (We focus on the biggest connected component for a couple of graphs).
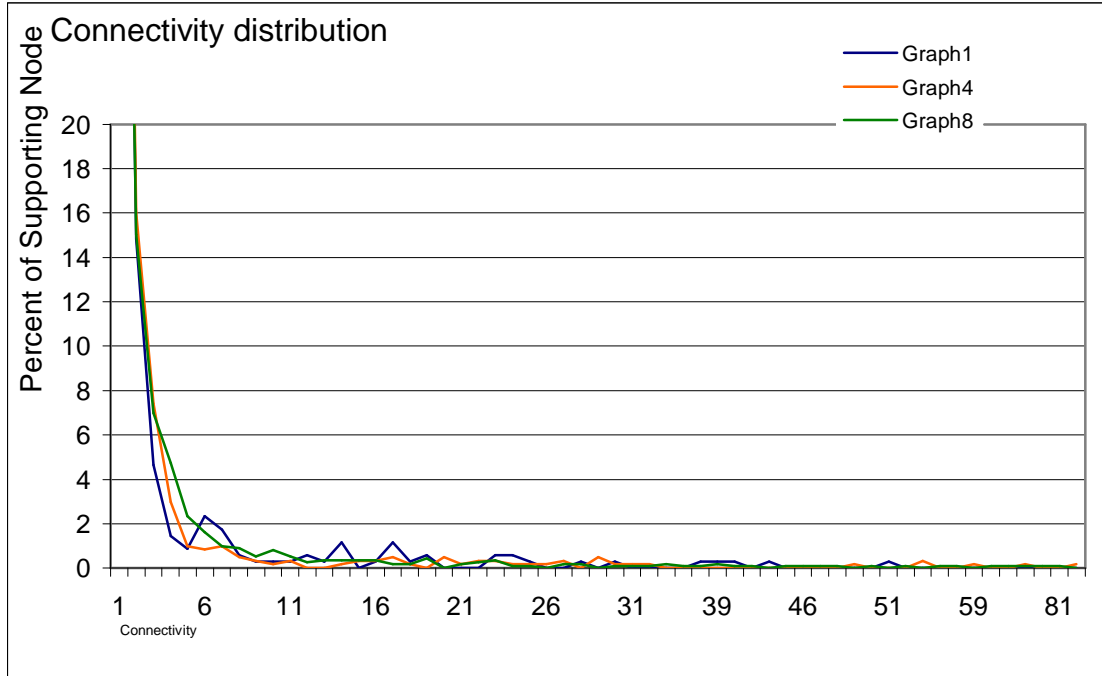
Figure 9: Connectivity distribution

The analysis of the connectivity implies that the connected component is made up of several high-degree nodes and a lot of low-degree ones.

4.6   Advanced connectivity analysis

This is what we had in our proposal: "we might be able to show that the network is less safe/reliable than claimed: following our approach an attacker could detect the 'hot spots' of the network (nodes bridging isolated subgraphs) and launch targeted attacks that will disconnect the network, overload or isolate the most interesting servers, or increase the network connectivity such that the network will die under the load a single query generates".

To identify 'hot spots' we propose the following robustness analysis: take out one node of the graph and see if the graph breaks into multiple components (in other words we verify if the graph is single connected and identify those points). We determine the number of connected components we break the graph into if we take the 'hot-spot' out.

Note that 'hot spots' are attacker's targets in Gnutella network, because if taken out of the network (actively or passively), the network will be separated to two or more distinct subpart, so these nodes badly affect the robustness of the Gnutella network.

Robustness Analysis of Graph5
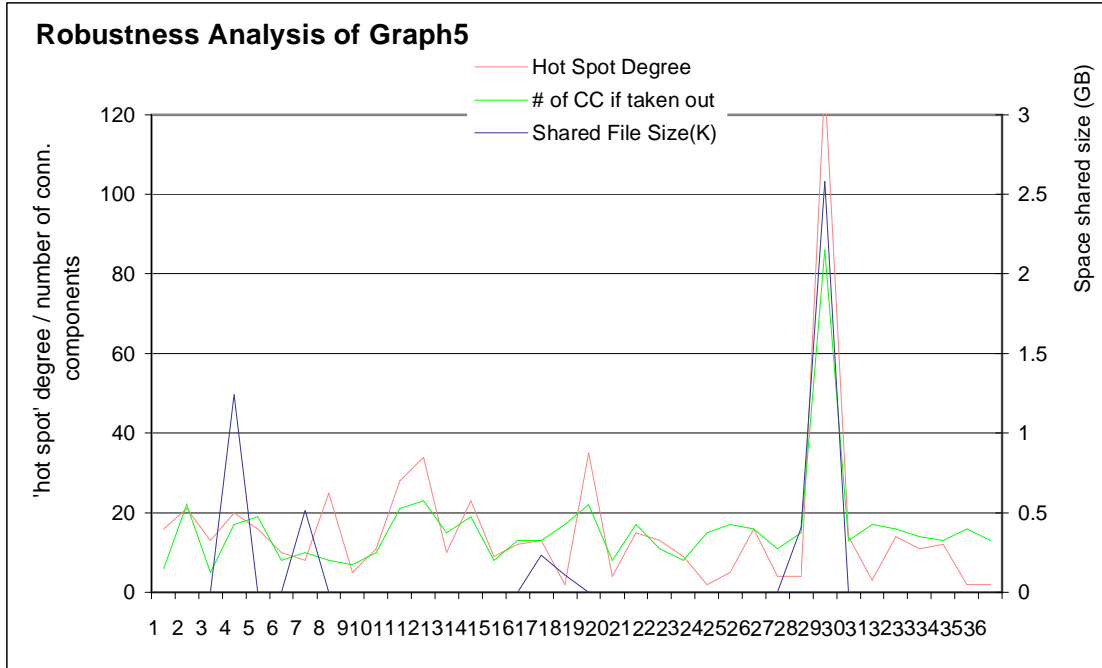
4.7 Usage patterns

We planed to test the following assumption: is there any relationship between the amount of information offered by a node and the connectivity of that node?

Whether the information offered by a node and the connectivity of it have some relationship also arouse our interest. But the analysis shows it might not be so easy to give a simple relationship. Other aspects of the networks contribute to the uncertainty of the relationship. More research work is still needed to explore it.

The next figure is a 2D plot of node connectivity (x axis) against the information shared by the node (y axis). We could group the points in 3 loose groups: nodes that don't share information but have a high number of connections; nodes that share a lot of files but have a small number of connections, and nodes that don't share information nor have connections. Almost all nodes are in this last category and they are essentially 'free riders'.
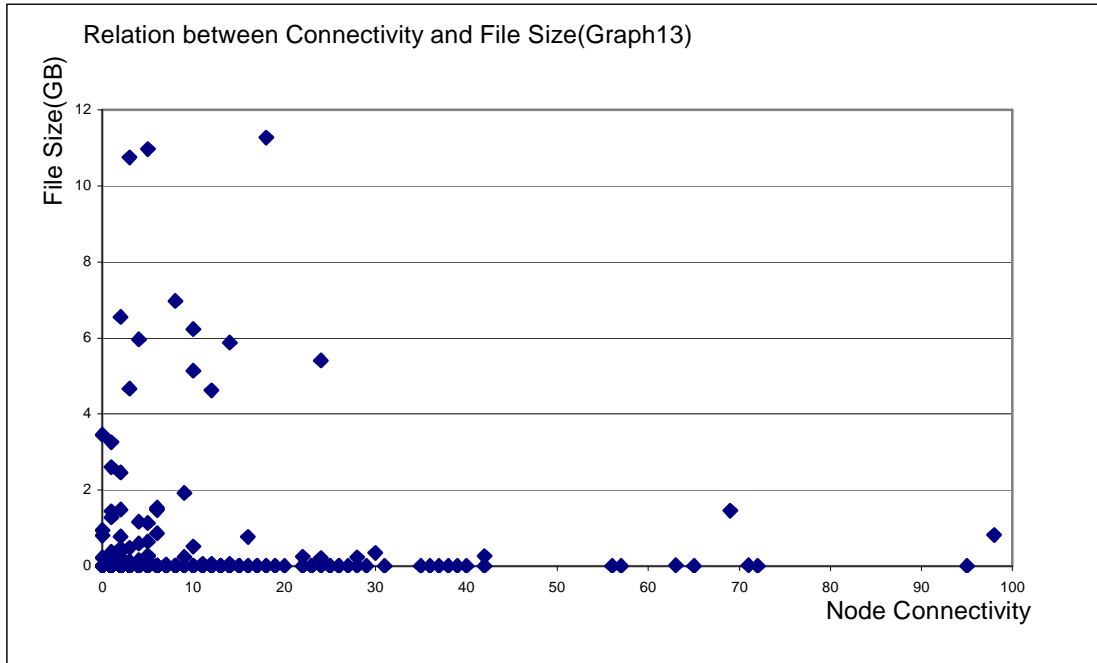
Figure 10: Usage patterns: data shared against node connectivity

This data, combined with our intuitions, suggests the following aspects of user behavior which influence network traffic and topology:

- Primary Use. Currently, MP3 audio (and some video) files are the most commonly sought and transferred files on the network. Since most of these files are copyrighted material, many users do not want to provide them due to legal problems. Also, these are typically large files, so if bandwidth is limited, then sharing will cause costs without benefits to the user. Hence the concentration most of the data in only a few nodes.

- Single Source For A Starting List Of Nodes. Most implementations automatically refer to a single source (like gnutellahosts.com) for a list of nodes to connect to. (Other implementations require the user to supply a list of starting nodes, which cannot easily be obtained except by finding a source which is a subset of the nodes listed by gnutellahosts.com) This tends to lead toward a very large connected component (good for data availability) and small diameter (good for data availability, bad for overhead traffic). jgnutella.org serves a similar function in Japan.

- Rapid Changing Of Neighbors. The rapid appearance/disappearance of nodes means that users will often switch their active connections (usually to maintain some fixed desired degree, but sometimes to aid the probability of successful search results). This is somewhat self-propagating; nodes that switch their neighbors disappear from the immediate environment, but at least are still connected to the graph as a whole.

○ MATEI'S SECRET GNUTELLA NETWORK OF SUPERCOMPUTERS IN ROMANIA ARE SLOWLY TAKING OVER THE WORLD!!

4.8 Generated traffic - empirical data and estimates

We try to evaluate the traffic one node has to serve. Obviously the more connections the *servent* has to the network the bigger the traffic that flows through the node. Given the flooding routing protocol for PING and QUERY messages the traffic the network generates is a linear function of the number of link in network topology.

Following the protocol description there are two types of messages that should make the largest part of the traffic. Those are the messages that are broadcasted (routed with a flooding scheme): PING and QUERY. The other messages should be simply routed back on the path the originator message has arrived. The PING traffic is pure overhead traffic – necessary to keep the network healthy in case of failures while QUERY messages traffic is user generate traffic.

We connected to the network with a fake *servent* and registered message type and length on the connection in order to get precise idea of the real traffic volume Gnutella generates. We randomly chosen a dozen of nodes to connect to and observed that results are similar. This is easy to explain: due to reasonably small network diameter and preponderant TTL=7 almost all messages reach all nodes.

We report here the number of queries per minute generated by each message type by host h24-71-121-229.ed.shawcable.net (measurements taken on December 7 from 0:28am to 6:43am, data is smoothed using a 3minutes trailing average):
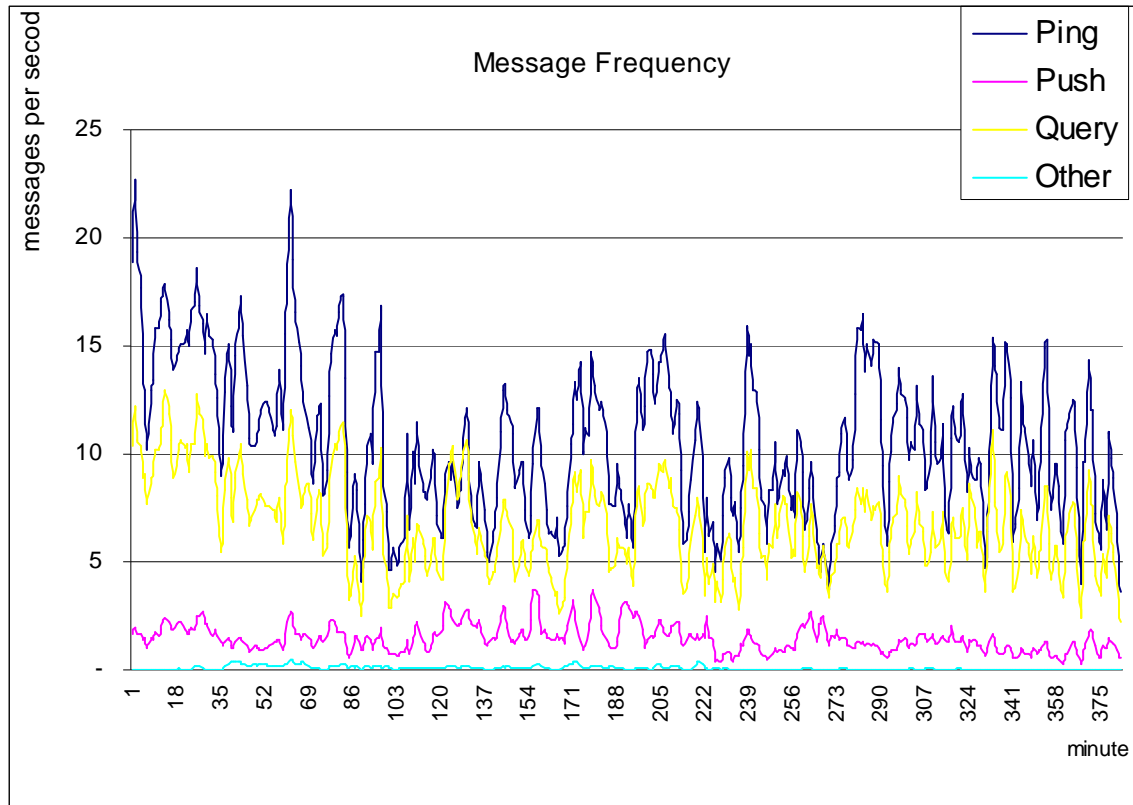
There are a couple of things that we observed during this experiment:

- A relatively significant (about 3%) number of bogus messages that wander through the network although they do not comply with the protocol: these messages either have different sizes than the protocol allows or they simply have an unspecified message type.

- There is a constant maximum length of 152 bytes for QUERY messages although the protocol does not specify any limit. This might be due to some servent implementations that try to avoid flooding attacks through limiting the message size.

- If we average the number of messages over a couple of minutes we observe that the number of messages generated is quite stable over time

As shown in the next Figure the system does not seem to make a very efficient use of resources: on average only 36% of the traffic is user generated traffic obeying protocol specification (QUERY messages). 55% of the traffic is purely overhead while 9% of the traffic contains either bogus messages (1%) or PUSH messages that are broadcasted by servents that are not fully compliant to the latest version of the protocol.
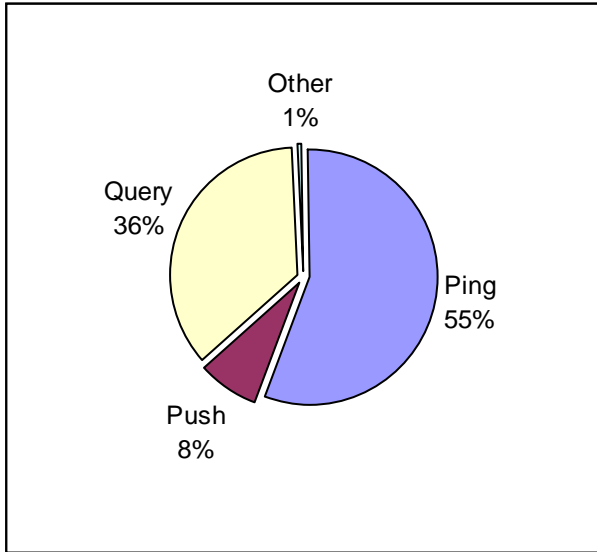
If we plot the volume of data transferred with each message type we observe a highly similar behavior. The only difference is that this time the volume of QUERY traffic is slightly larger (41%) as the average QUERY message is 35 bytes long while a PING message is 23 bytes long.

Considering a 40 bytes TCP/IP headers overhead per message and ignoring the overhead brought by lower network layers the next plot presents the overall volume of data transferred (in Kbps) averaged over one minute intervals (general average for our 6:15 hours measurement is 11.5Kbps).
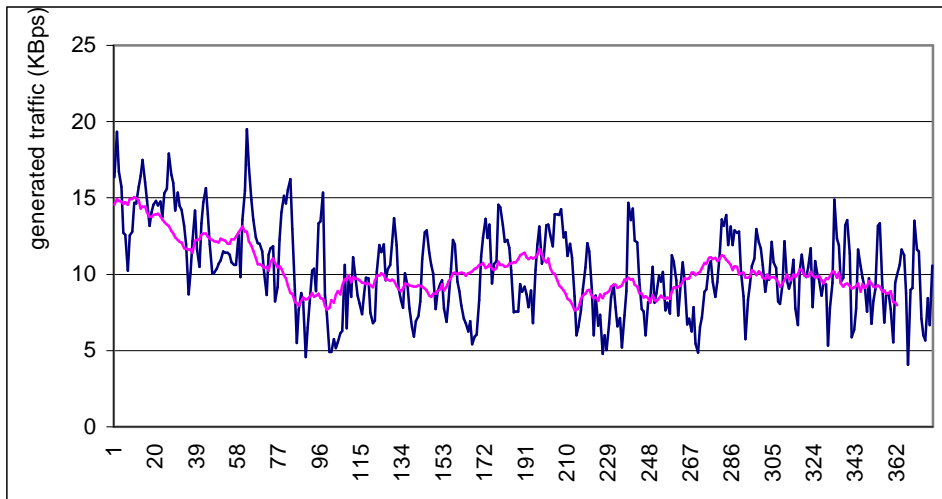
In view of this result a node can adjust the number of connections it can operate according to the link bandwidth available. As a rule of thumb:

$$NoOfConnections = \frac{Link\ Bandwidth}{15Kbps}.$$

(We have used a 20% margin that should give the node enough room to accommodate traffic peaks).

4.9   Level of noise in the network, estimate the number of faulty servents

One of the surprises we had while looking carefully at crawling data was to see a large number of instances where servents were fully compatible with the protocol:

A servent is supposed to answer a PING with only one PONG. Routing nodes are responsible to filter out duplicate messages if they ever appear. After sending a PING message our crawler was often (about 10%) flooded with duplicate PONG replies. This might be due to a bug in the routing mechanism of some servlents or/and a bug in the PONG reply mechanism or an external attack.

We observed a high number of hosts that were reported as members of the network although we could not contact them. The next figure details the reasons for which we couldn't contact those servers. For most of them it is probable that the crawler tried to investigate them after the node left the network. However there was a significant percent of hosts that did not follow Gnutella's peer to peer connection establishment.
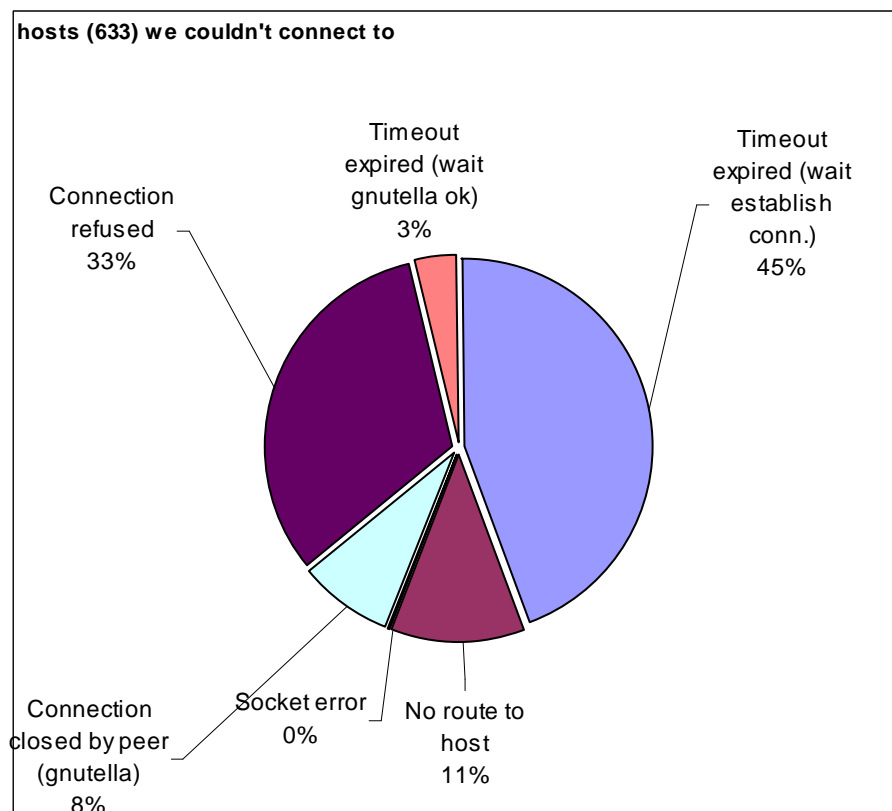
There is a class of servents not obeying the protocol that is particularly worrying (especially since they are as many as about 20% of all nodes we find). These servents report fake contact info (could be reserved IP addresses, private IP addresses, port declared 0 and the list could continue). These hosts have a connection open to somebody in the network (an therefore they benefit of all network resources) but report a fake contact address, therefore nobody will ever contact them.

We believe this is another face (a more dangerous one this time) of the 'free riding' behavior observed by [1]. [1] reports (and we confirm) that only a small part of the nodes have data available while the majority do not share any data and are merely using the service. However these nodes do provide a service to the community: they have a key role in keeping the network connected (by answering PING messages and letting other servers to connect to them). Nodes that do not disclose their IP addresses (together with nodes that don not accept any incoming Gnutella connection) do refuse to provide even this minimal service to the Gnutella network.

Another explanations could be that users believe that not disclosing their IP address would protect their anonymity (this is false but we will not elaborate further on this) or that the network is under another type of attack.

4.10 Domain analysis

We want to know where of each node in Gnutella network comes from. It arouses our interest because this analysis will enable us to look insight into the scalability of the network.
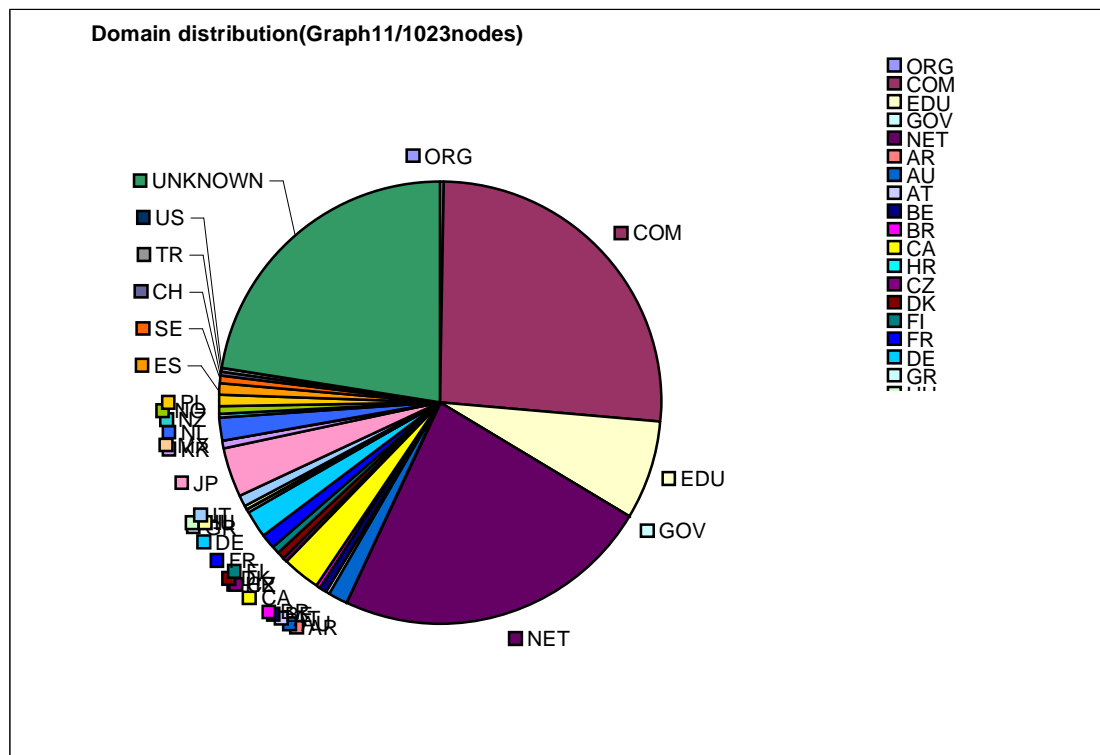
In a large-scale survey, we see the US users takes the biggest share of the whole network, but at least one out of four network hosts was found outside the US-centric top-level domains, with Germany and Japan dominating the non-US population. Gnutella network is truly an international phenomenon.

5.   Attack Scenarios

RIAA and other copyright agencies are suspected of attempting to sabotage the network (*what is legality of network attack*?). They already have a means to order servers to stop distributing material they own (NET act), but it is cumbersome and difficult in decentralized networks. Other groups or individuals may attack Gnutella networks for other purposes as well.

- o   Due to the concentration of data in a few nodes, however, it may be easy for them to stop these "servers" legally, or by overloading / isolating them: Denial of Service attacks – unbounded queries (high-TTL queries for random data), flooding the network with (large) flood-routing overhead messages (forces slower nodes to drop packets and possibly partition the network), non-Gnutella attacks on points of high connectivity or file availability. Many of these attacks (intentional or not) are weakened by servant implementations which refuse to propagate large messages, too-frequent PINGs, etc.

- o   False "servers" may respond to all queries, either forcing searchers to wait for a network timeout after issuing a GET, or by returning large amounts of useless data. This is expensive to implement, but RIAA is wealthy and it is easy to frustrate someone who spends hours downloading a large and worthless file.

- o   Graph-structure attacks may be attempted – either by trying to greatly increase or decrease connectivity within the graph. Increasing connectivity tends to cause smaller total diameter, which increases overhead traffic significantly. Less connectivity means the network may be partitioned into several connected components, decreasing data availability.

6.   Where does Gnutella succeeds/fail?

Succeed:

- o   The protocol is extremely SIMPLE.

- o   It works well in a highly dynamic environment.

- o   A lot of data is shared, there are a lot of concurrent users

- o   Nodes join in and leave smoothly

- o   It manages to integrate diverse 'servent' implementations. (Drawback – it is difficult to enforce ideal behavior)

Fail:

- o No anonymity.

- o Easy to attack (previous section details a couple of attack scenarios)

- o Does it use resource efficiently? We doubt!

- o Doesn't scale well

- o Network tends to become centralized around data providers

- o Does not prevent "free riding"

7. Possible protocol improvements

Gnutella's protocol needs more efficient use of available resources in order to be scalable.

Some speed improvements can be obtained simply by eliminating vulnerabilities to attack. Reducing overhead will reduce disconnections and dropped packets. Apart from this, increased connectivity (esp. adding nodes to bridge connected components) helps preserve communication despite loss of nodes.

We identify two possible strategies to modify the protocol:

1. Optimize the graph but not change (significantly) the protocol. This basically implementing smarter servents within the current framework.

- o Based on our analysis we might be able to come up with possible improvements in servents behavior such that in the case the graph structure is not optimal. An optimal network is one where the diameter is minimized (to reach as many nodes as possible with a query) while the number of edges is also minimized (to reduce overhead traffic).

- o For example if we discover the graph is single connected or disconnected we could suggest a modification of the protocol to force increased connectivity. Dynamic TTL for messages based on graph topology should allow performing queries at a specified cost.

- o Mention Reflector (but this starts to make the network hierarchical)

- o Sanity Workers – Some (trusted) automated crawlers could analyze topology data and tell nodes to add or drop links in order to optimize the graph. This assumes trust of the sanity workers (could be done by encryption) and also assumes that they can crawl the network fast enough (without introducing too much overhead!) to make timely changes. Of course, they are needed only in situations where the network retains inefficiency for extended time periods.

2. Change the protocol completely:

- o The first place to start would be to get rid of the expensive flooding routing. FreeNet already does this using a combination of depth first search and routing based on query content.

- Do not route query replies and PONGs on the same path (this way you sacrifice some anonymity – but it's fairly insecure anyway)

- Add in caching mechanisms (again this will drive the protocol closer to FreeNet)

- Anonymity issues: Direct file transfers mean anybody receiving a query can learn the IP by pretending to have the files in question. So TTL<<diameter means less nodes receive your query. Otherwise, only FreeNet is able to effectively do this. There is no auto replication in Gnutella.

8.   Conclusions

Simple protocol specifications, loose enforcement of rules, and varying user motivations combine to create a complex 'natural selection" environment. Implementations, methods of dealing with improper behavior, and user behaviors evolve according to demands the larger network. Arguably, this method of "developing" a P2P network works bottom-up, by placing a loose specification (with sufficient rewards for adopters of the protocol) online and allowing a disorganized response to problems, is more conducive to tackling the complexities of P2P networking than a strict top-down approach.

Gnutella assumes a fairly hostile environment: privacy is an issue, and nodes are assumed to be impermanent. Such a protocol may be useful in other hostile environments, such as battlefield communications, lightweight wireless devices with short range and low power transmission.
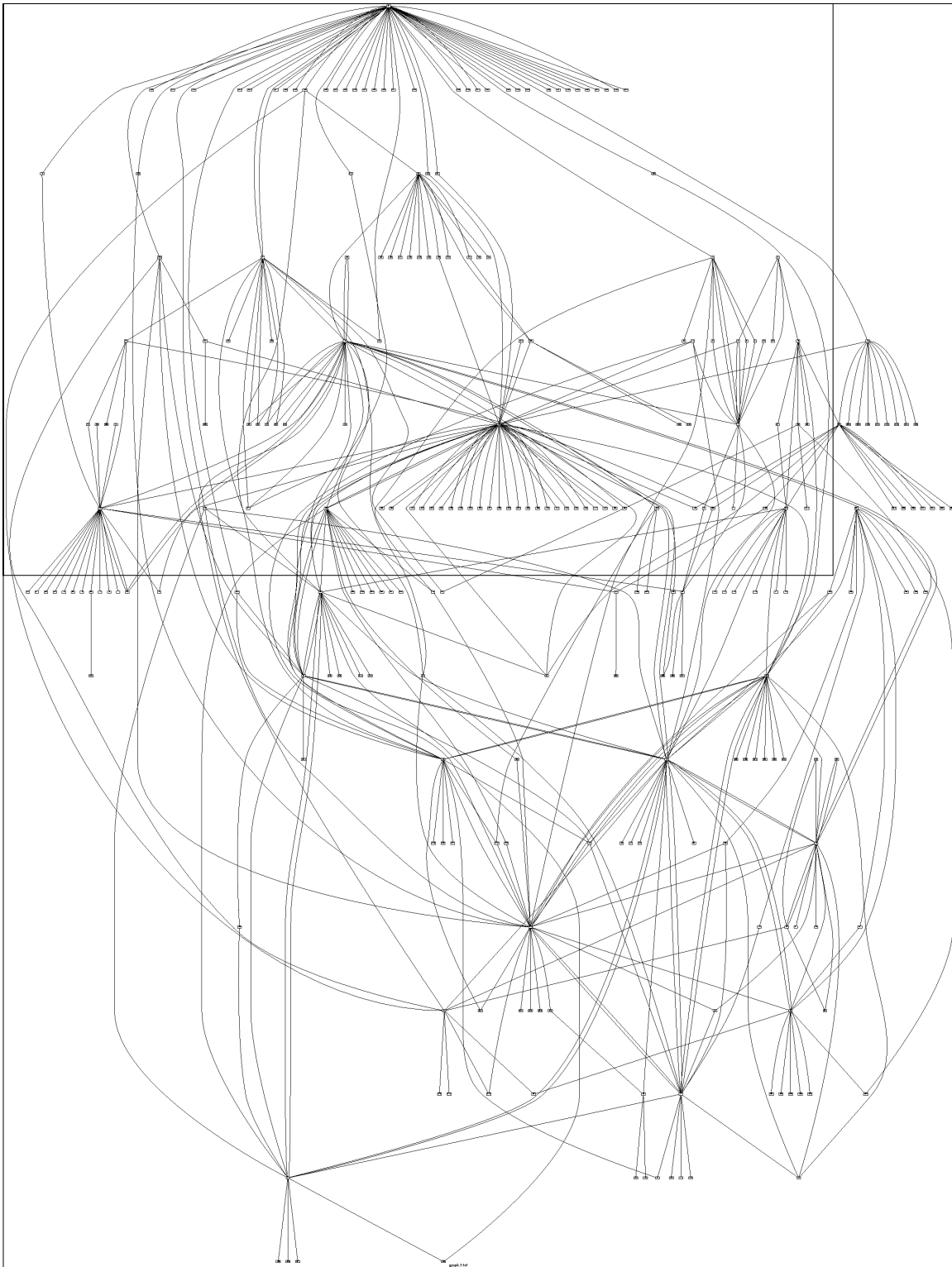
The improvements we suggest fall into two categories: solutions preserving the protocol (Reflector, Sanity Workers, etc) and changes that generally sacrifice anonymity or survivability in exchange for performance. However, the topological maintenance methods we described may be able to preserve survivability even better than the existing protocol, and anonymity in Gnutella is currently weak anyway.

9.   References

[1] – Free riding on Gnutella, *Eytan Adar and Bernardo Huberman*, First Monday Volume 5, Number 10 - October 2nd 2000, http://www.firstmonday.dk/issues/issue5_10/adar/index.html

[2] – http://www.gnutella.co.uk

[3] – Webopedia (Online dictionary and search engine for computer and Internet technology). http://www.webopedia.com/

[4] – The Gnutella protocol specification v4.0 - http://dss.clip2.com/GnutellaProtocol04.pdf

[5] – Gnutella: To the Bandwidth Barrier and Beyond, *dss.clip2.com*, November 6, 2000
http://www.gnutellahosts.com/gnutella.html#s2

[6] - Bandwidth Barriers to Gnutella Network Scalability, dss.clip2.com, September 8, 2000 http://www.gnutellahosts.com/dss_barrier.html

10. Appendix 1

graph.3.hcl

---

[1] http://www.groove.net/