Analysis of the Traffic on the Gnutella Network

Kelsey Anderson
University of California, San Diego
CSE222 Final Project
March 2001
<*kelsey@cs.ucsd.edu*>

Abstract

The Gnutella network is an overlay network developed as a peer−to−peer file sharing system which uses the Gnutella protocol. The Gnutella network was designed by Nullsoft in 1999 and has quickly grown in popularity since its release. Unfortunately this growth has occurred rapidly and haphazardly and there has been little research done into the performance and workload characteristics of Gnutella. This paper describes some initial analysis of the traffic on the Gnutella network.

Introduction

The Gnutella network is a peer−to−peer overlay network which utilizes the Gnutella protocol to handle queries and maintain connections with other hosts in the network. A peer−to−peer network is characterized by the ability of hosts on the network to act simultaneously as both servers providing data and services to others and as clients using data and services. In Gnutella, these hosts applications have been termed *servents*, or alternately, *gnodes*. A software application implementing the Gnutella protocol was developed and briefly released in 1999 by Nullsoft, a subsidiary of America Online, and was intended to be a peer−to−peer file sharing system. The software was quickly made unavailable, but not until several thousand interested software developers had downloaded the client. Afterwards, the Gnutella protocol was reverse engineered and documented, resulting in many different implementations of the servent application and an ad hoc network began growing. Since then, the popularity of the Gnutella network has grown tremendously since its development due to the legal battles surrounding Napster, the commercial peer−to−peer digital music sharing network. The decentralized nature of the Gnutella network and the open availability of the protocol specification has made Gnutella a natural candidate for Napster−like file sharing to continue.

The growth of the Gnutella network has not been without problems, however. In August of 2000, the Gnutella network encountered its first obstacle. The bandwidth required to transmit Gnutella network traffic became sizable enough to prevent users connected via modem from participating in the network. This problem was documented by Clip2DSS and a solution was quickly crafted which provided modem users a buffer from others with faster network connections [1]. This anecdote illustrates a primary problem which still plagues the Gnutella network today − lack of information and data about how the network is operating. To date there has been little comprehensive analysis of the behavior of Gnutella and many open questions remain. This paper describes some initial results of an ongoing investigation into the characteristics of the Gnutella network. Section 1 discusses the Gnutella protocol. Section 2 describes other research that has been done on Gnutella. Finally, section 3 discusses the traces, traffic analysis, and

results obtained from this work, followed by future research directions and concluding remarks.

1. The Gnutella Protocol

The Gnutella protocol defines how servents communicate in the Gnutella network. The canonical protocol specification for the Gnutella protocol was developed through reverse engineering and has been documented by Clip2DSS [2]. There are five types of packets used to communicate:

- *Ping* − used to actively discover other servents on the network
- *Pong* − a response to a Ping that also tells how much data is being shared and the network address (IP address) of the responding gnode.
- *Query* − the mechanism for searching for data on the Gnutella network in a distributed fashion.
- *QueryHit* − a response to a Query that tells what data matches the query and the IP address of the servent where the data can be found or downloaded.
- *Push* − a mechanism that allows a servent behind a firewall to forward files to other servents.

Each Gnutella packet contains a header with the following fields:

- *Query ID* − a 16−byte unique query identifier used to correlate Queries and QueryHits.
- *Packet Type* − describes what kind of packet is sent. The valid descriptors are:
  0x00 for Ping;
  0x01 for Pong;
  0x40 for Push;
  0x80 for Query;
  0x81 for QueryHit.
- *Time To Live (TTL)* − the number of hops that the packet will be forwarded to other servents.
- *Hops* − the number of times the packet has already been forwarded.
- *Payload Length* − the size in bytes of the data in the packet, not including the header. For Ping packets this is always zero bytes; all others have varying payload sizes.

Gnutella servents join the network by connecting to one of several commonly available servents running in the network on a specified port. The most common one is gnutellahosts.com:6346, which is run by Clip2DSS [3]. The servent returns a list of recently confirmed active servents using the Ping and Pong mechanism, which the local servent then uses to establish connections throughout the Gnutella network. Servents also keep a configuration file of all other servents encountered on the network, which can be used to connect again. The transient nature of many servent connections makes the Gnutella network especially dynamic so that measurements of the system always reflect the changing topology and content of the currently connected servents. This property also adds complexity to the task of measuring and interpreting data from the network and contributes to the lack of relevant knowledge about the current state of Gnutella.

Once a servent is connected to other active servents in the Gnutella network, packets are forwarded using a broadcast mechanism between gnodes. When a servent receives a Ping or Query with a positive−valued TTL, it will decrement the TTL value in the header, increment the Hops value, and forward the packet to all connected servents except for the one from which the packet originated. The servent will also formulate a Pong in reply to a Ping (or a QueryHit in reply to a Query if applicable) and send it to the originating servent. The Query ID for an active query is also cached so that

QueryHits from other servents can be properly forwarded back to the originating servent. This establishes a communication model based on *swamping* [4], which minimizes the time needed to fulfill a query but at the expense of network resource consumption. Fortunately downloads of content that satisfy a user's query are performed out−of−band using direct servent−to−servent connections via HTTP, so that further resources are not consumed in the Gnutella network for file transfers.

Many of the critical properties that affect the network performance are either set by the user of the servent or have values that differ by implementation. For example, the number of connections to other servents in the network is a user−configurable property that impacts the amount of network bandwidth required to interact effectively in the Gnutella network. Many servent implementations default to one connection for modem users and four connections for users with higher bandwidth links, but this is only a recommendation based on experiential evidence of questionable age. Also, the value assigned to the Time To Live field in the originating servent affects how long the packet will exist in the network, how many other servents will process the query, and possibly how many results are returned from a query. Many implementations have a default TTL value of 7 or 9 hops, but some users have changed this property setting to reflect a wider network traversal. More information about specific implementations can be found at [5].

2. Related Gnutella Research

There has been remarkably little research done to characterize the performance or workload of the Gnutella network. In addition, rapid growth, the dynamic nature of the network topology, competing implementations of the servents, and a lack of a central standardizing body for Gnutella has all contributed to make older research nearly irrelevant. In fact, perusing the Clip2DSS website, [3], or O'Reilly's Peer−to−Peer technology website, [6], will yield much of the published research about Gnutella. A more detailed investigation will yield a few recent works−in−progress.

The most prominent research done by Eytan Adar and Bernardo Huberman at Xerox PARC analyzed Gnutella network traffic for patterns in file sharing [7]. The authors found that nearly 70% of users on the network shared no files, and that nearly half of all queries were fulfilled by 1% of the hosts that shared any files. The authors claimed that such "free riding" led to network performance degradation and vulnerability to collapse. This paper sparked much controversy in the Gnutella community which resulted in immediate changes made to most servent implementations to discourage free riding by requiring users to share files in order to make queries on the network and by caching any downloaded files resulting from queries as shared files. Many Gnutella users now use the updated versions of the servents and it is unknown whether the claimed risk of performance degradation still exists on the network.

Steven Bellovin at AT&T Labs Research investigated security in the Gnutella network and protocol and discussed his preliminary work in a talk on June 21, 2000 [8]. He found that the Gnutella protocol had several characteristics that made security a concern for users, including the ability to spoof response addresses, advertise inaccurate content and topology information, and trace the Query ID to a particular user. In addition, he determined that many implementations of Gnutella servents were

prone to have bugs like buffer overflows that could make servents misbehave and impact the performance of the network.

In addition, a number of people have looked at network topology, scalability, and reachability in the Gnutella network. Steve Gribble and Stefan Saroiu at the University of Washington modified a servent to crawl the network and collect topological information [9] and display the current network topology in a visual format. A group at the University of Cincinnati have also implemented a topological crawling servent and used their data to analyze the scalability of the Gnutella protocol and host reachability within the network [10] [11] [12]. And finally, Jordan Ritter, a founding developer of the Napster network, performed a numerical analysis of the scalability of Gnutella using the most common values for parameters like number of open connections per gnode [13]. He claims that the Gnutella network will never scale successfully because of the broadcast nature of the Gnutella protocol.

Clip2DSS has performed some initial analysis of Gnutella network traffic as well. They recorded traces of packets on the Gnutella network in August and September 2000 and performed some analysis on number of gnodes on the network, types of queries made by content type and file extension, and geographic distribution of gnodes on the network by country, top−level domain, and service provider [1]. Their work provided the first data about the performance and content available on the Gnutella network. However, with the apparent growth rate of the Gnutella network, this data may be dated.

In order to pursue some research in topological self−sorting algorithms for gnodes in the Gnutella network, a more thorough analysis of traffic patterns is required. This paper describes some initial analysis of Gnutella traffic that will be used to gain further understanding about the Gnutella network characteristics.

3. Traces and Traffic Analysis

In order to collect packet traces of Gnutella traffic through a single node, I instrumented a Gnutella servent to save packet contents to a file. The servent implementation Gnut version 0.4.20 provided well−documented source code written in C and seemed to be the most developed open−source servent available [5]. Several traces were taken during the week of 3/12/2001, but this analysis will focus on the trace taken during 3/16−3/18 that covers approximately a 35−hour period. The Gnut servent ran on an Active Web cluster node at UCSD, which is an Intel−based machine with 2 600MHz processors and 512MB of memory, running Solaris 8. The traces were filtered using handcoded Tcl scripts that parsed out particular fields of interest and graphs were generated using Microsoft Excel and StarOffice 5.2.

Some initial questions were posed:

- How far away (in network hops) are users making queries and users fulfilling queries?
- What kinds of queries are being made, how often are queries fulfilled, and how many query responses are returned?
- How many QueryHits are returned by a single gnode and how many gnodes respond to a query?
- How much traffic of each type does each servent handle?
- How much data is available on the network and how many gnodes are currently reachable?

The traces resulted in data that answers some of these questions. The traces showed that Pings and Pongs were just over 50% of the traffic seen on the network, and that Queries were another 40% of packets seen in the traces. Figure 1 shows the breakdown of traffic by packet type.

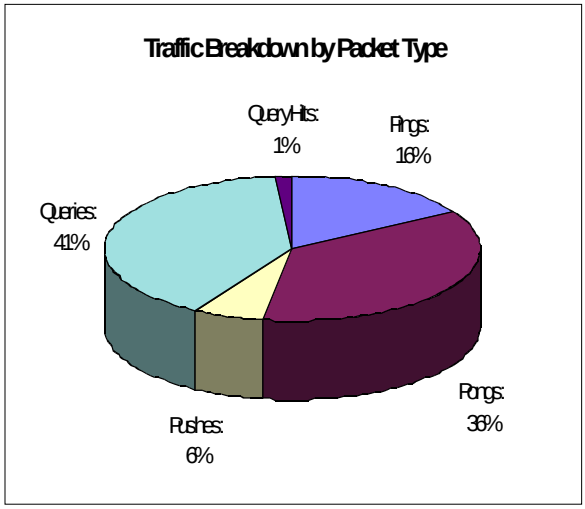

Figure 1: Traffic Breakdown by Packet Type

The figure also shows that QueryHits make up only 1% of the traffic seen by a single gnode in the network, and that Pushes make up a surprising 6% of packets seen. This analysis points to a high overhead of traffic used to find other hosts and maintain network connectivity that would be detrimental to the scalability of the network.

The next analysis compared the proportion of unique Queries (determined by the unique query ID stored in the packet header) that have hits returned through a single gnode relative to all Queries. Figure 2 shows that only 3% of Queries have hits that are returned on a particular path through a single gnode.

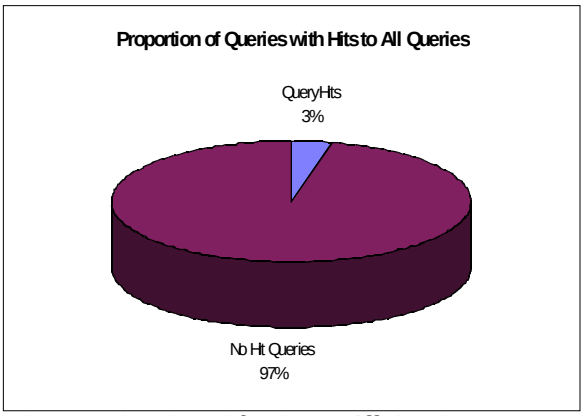

Figure 2: Queries with Hits vs. All Queries

It is unclear why so many queries are unsuccessful on a particular path. Further work needs to be done to distinguish why some queries were successful, why most are not, and what can be done to improve the chances of having a successful query.

Figure 3 shows the distribution of how many responses were returned per successful query along the path with the tracing gnode.

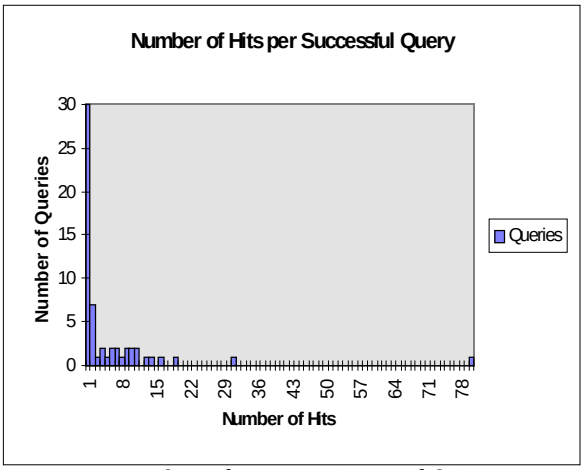

Figure 3: Number of Hits per Successful Query

This graph shows that the number of responses received by a successful query ranged from 1 to 80 through the tracing gnode, with most queries obtaining only a small number of hits. Further analysis remains to determine what types of queries result in large numbers of responses.

The rest of the analysis focused on *hit distance*, or how far apart (in network hops) were the servents that queried and responded to a particular query, and other questions about reachability. Figures 4 and 5 show the distributions of Hops and TTL values extracted from all packets. The TTL values are a user−settable parameter in most servent implementations, so this gives an indication of how far users want their queries to travel.

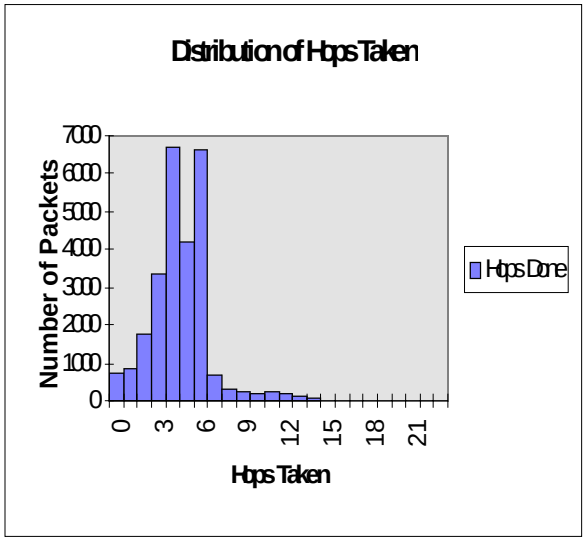

Figure 4: Distribution of Hops for all Packets

The Hops graph shows that most packet had traveled 6 or fewer hops when recorded by the traces, but that some packets had traveled 20 hops or more. The packets shown with zero hops originated from neighboring nodes since the hopcount is changed upon forwarding the packet in Gnut. The TTL graph shows some surprising data. Most packets had TTL values of 15 or less, but nearly equal numbers of packets were seen for each value. In addition there were significant numbers of packets with TTL values around 20, which indicates that users have been trying to cast their queries in a wider range to possibly obtain more query responses. It is unclear whether this is a successful tactic for increasing the chances of getting responses and what the impact of wider packet distances may be on Gnutella network performance.

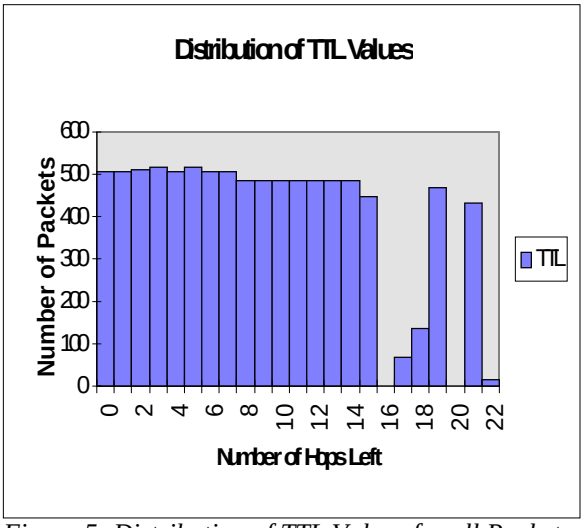

Figure 5: Distribution of TTL Values for all Packets

Figure 6 shows the distribution of hop counts taken from only the Queries.

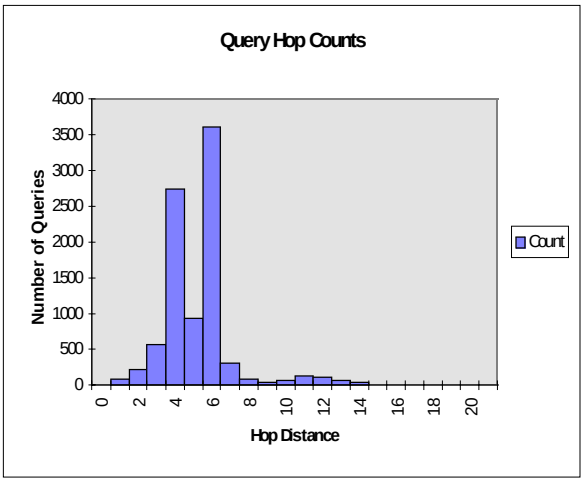

Figure 6: Distribution of Hops for Queries

Most queries had traveled 4 or 6 hops before they were recorded by the tracing gnode, but some had traveled much further. This graph is particularly interesting because of the bimodal distribution of hops taken and because of the wide range of hop counts

seen for queries. Further analysis remains to determine whether the bimodal structure is due to differing preset parameter values for the most commonly used servent implementations or whether it is due to some quirk of the network topology.

Finally, figure 7 shows the hop count distribution for QueryHit packets only. This graph has a much smaller range with most QueryHit packets traveling 1 or 4 hops, and a few packets traveling a maximum of 6 hops before being recorded.

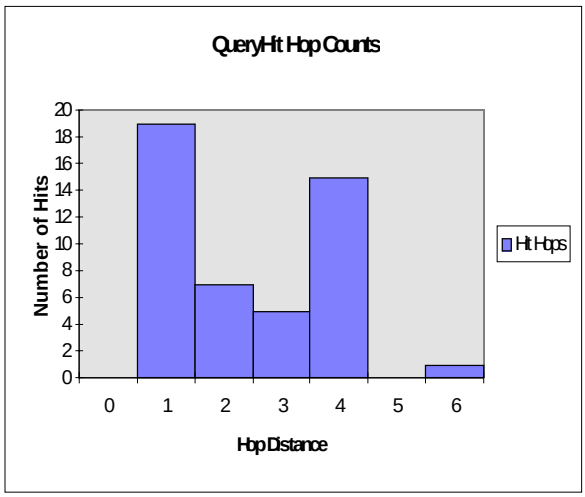

Figure 7: Distribution of Hops for QueryHit Packets

This graph also shows a bimodal distribution, like the Query distribution, but over a much smaller range.

Conclusion

This traffic analysis of the Gnutella network provided insight into some of the questions described previously, but also created more questions to answer. These traces only showed the Gnutella network from the perspective of a single gnode, so the view is somewhat limited and the ability to correlate queries and responses is constrained. It would be interesting to instrument several cooperating Gnutella servents and place them in disparate parts of the network and see how their views differ. The existing traces were also taken passively by recording only those packets that were forwarded from neighboring gnodes. Another future experiment would be to actively make queries at locations around the network and record the responses from different servents. Finally, it is clear that the Gnutella protocol and network face some challenging scalability issues in order to succeed in the future. My long−term research objective is to improve the scalability of the Gnutella network while not hindering gnode and content reachability through implementing self−organizing algorithms. Gnutella, and other instances of peer−to−peer networks, offer the promise of exciting new applications that benefit users in ways that previous networks have not. Only through understanding the organizing principles of networks like Gnutella will we find their true value.

References

[1] http://dss.clip2.com/gnutella.html

[2]http://dss.clip2.com/GnutellaProtocol04.pdf

[3] http://www.clip2.com

[4] Harchol−Balter, Mor, Tom Leighton, and Daniel Lewin. *Resource Discovery in Distributed Networks.* Proceedings of the ACM, PODC, Atlanta, GA, May 1999.

[5] http://www.gnutelliums.com

[6] http://www.openp2p.com

[7] Atar, Eytan, and Bernardo A. Huberman. *Free Riding on Gnutella.* Xerox Palo Alto Research Center, October 2000. http://www.parc.xerox.com/istl/groups/iea/papers/gnutella/Gnutella.pdf

[8] Bellovin, Steven M. *Security Aspects of Napster and Gnutella.* AT&T Labs Research talk, June 2000. http://www.research.att.com/~smb/talks/NapsterGnutella/index.htm

[9] Gribble, Steve, and Stefan Saroiu. The Snowtella Project. http://www.cs.washington.edu/homes/tzoompy/snowtella
[10] Jovanovic, Mihajlo A., Fred S. Annexstein, and Kenneth A. Berman. *Scalability Issues in Large Peer−to−Peer Networks − A Case Study of Gnutella*. http://www.ececs.uc.edu/~mjovanov/Research/paper.html
[11] Annexstein, Fred S., Kenneth A. Berman, and Mihajlo A. Jovanovic. *Latency Effects on Reachability in Large−scale Peer−to−Peer Networks and a Case Study of Gnutella*. http://www.ececs.uc.edu/~annexste/gnat/gnutab.html
[12] Jovanovic, Mihajlo A. *Analyzing and Modeling of P2P Networks: Case Study on Gnutella*. http://www.ececs.uc.edu/~mjovanov/Research/gnutella.html
[13] Ritter, Jordan. *Why Gnutella Can't Scale. No, Really*. Personal Communication. <jpr5@darkridge.com>