Trace Driven Analysis of the Long Term Evolution of Gnutella Peer-to-Peer Traffic*

William Acosta and Surendar Chandra

University of Notre Dame, Notre Dame IN, 46556, USA
{wacosta,surendar}@cse.nd.edu

Abstract. Peer-to-Peer (P2P) applications, such as Gnutella, are evolving to address some of the observed performance issues. In this paper, we analyze Gnutella behavior in 2003, 2005, and 2006. During this time, the protocol evolved from v0.4 to v0.6 to address problems with overhead of overlay maintenance and query traffic bandwidth. The goal of this paper is to understand whether the newer protocols address the prior concerns. We observe that the new architecture alleviated the bandwidth consumption for low capacity peers while increasing the bandwidth consumption at high capacity peers. We measured a decrease in incoming query rate. However, highly connected *ultra-peers* must maintain many connections to which they forward all queries thereby increasing the outgoing query traffic. We also show that these changes have not significantly improved search performance. The effective success rate experienced at a forwarding peer has only increased from 3.5% to 6.9%. Over 90% of queries forwarded by a peer do not result in any query hits. With an average query size of over 100 bytes and 30 neighbors for an ultra-peer, this results in almost 1 GB of wasted bandwidth in a 24 hour session. We outline solution approaches to solve this problem and make P2P systems viable for a diverse range of applications.

1 Introduction

In recent years, Peer-to-Peer (P2P) systems have become popular platforms for distributed and decentralized applications. P2P applications such as Gnutella[1], Kazaa [5] and Overnet/eDonkey [6] are widely used and as such, their behavior and performance have been studied widely [11,3,8]. Gnutella, although popular, has been shown to suffer from problems such as free-riding users, high bandwidth utilization and poor search performance. With the number of users in the network growing and the ultra-peer architecture becoming more dominant [9], solving the problems of bandwidth utilization and search performance becomes increasingly important to the long-term viability of Gnutella as well as other filesharing P2P applications.

In this paper we present an analysis of the trends in Gnutella traffic 2003, 2005, and 2006. First, we examined large-scale macro behaviors to determine how

* This work was supported in part by the U.S. National Science Foundation (IIS-0515674 and CNS-0447671).

S. Uhlig, K. Papagiannaki, and O. Bonaventure (Eds.): PAM 2007, LNCS 4427, pp. 42–51, 2007.

Gnutella traffic evolves over the long term. We captured traffic in 2003, 2005, and 2006 and compare the message rate, bandwidth utilization, and queueing properties of each Gnutella message type to determine the changes in characteristic behavior of Gnutella traffic over the long term. We found that the percentage of the bandwidth consumed by each message type changed considerably. We also studied more localized behavior such as the query success rate experienced by peers forwarding queries into the network. We found that the overwhelming majority ($> 90\%$) of the queries forwarded by a peer do not yield any query hits back to the forwarding peer. We show that earlier changes to the Gnutella protocol had not achieved the intended benefits of alleviating poor search performance and high bandwidth utilization.

2 Gnutella Overview

First we describe the Gnutella system. Gnutella is a popular P2P filesharing application. Gnutella users connect to each other to form an overlay network. This overlay is used to search for content shared by other peers. The Gnutella protocol is a distributed and decentralized protocol. Peers issue request messages that are flooded to all of their neighbors. Each neighboring peer, in turn, forwards the request to all of its neighbors until a specified time-to-live (TTL) is reached. This flooding mechanism allows for reaching many peers, but can quickly overwhelm the available network bandwidth. Next we describe the two major versions of the Gnutella protocol.

2.1 Protocol v0.4

The original Gnutella protocol, v0.4, assumed that each Gnutella peer was equal in terms of capacity and participation. The v0.4 protocol specified four major messages: Ping, Pong, Query, and QueryHit. The protocol also specified other messages such as Push requests, but these messages were rare and did not represent a significant percentage of the messages sent in the Gnutella network. The Ping and Pong control messages are used by Gnutella clients to discover other peers in the network. Ping and Query messages are flooded to each neighbor. The flooding continues until the TTL for the request has been reached. Responses such as Pong and QueryHit messages are routed back to the originator along the path of the request. When a peer receives a Query message it evaluates the query string and determines if any of its shared content can satisfy the query. If so, the peer sends back a QueryHit message along the path the it received the Query. Each QueryHit message contains the following information: the number of objects that match the query, the IP address and port of the peer where the objects are located, and a set of results including the file name size of each matching object.

2.2 Protocol v0.6

As Gnutella became popular, both the size of the network and the amount of traffic on the network increased. The increase in network traffic coupled with

the poor Internet connections of many users [11] overwhelmed many peers in terms of routing and processing messages. In order to overcome the performance limitations of the original protocol, the v0.6 [2] was introduced. The new v0.6 ultrapeer architecture reduced the incoming query message rate thus lowering the bandwidth requirements to process queries. However, as we show in this paper, the architecture change did not eliminate the problem of high bandwidth consumption; it shifted it to a different place in the network. Specifically, the v0.6 protocol requires the ultrapeers to maintain many connections.

3 Related Work

Saroiu et. al. [11] and Ripeanu et. al. [10] studied several aspects of the Gnutella file-sharing network. The authors discovered that new clients tended to connect to Gnutella peers with many connections. This led to an overlay network whose graph representation had a node degree distribution of a power law graph. The authors also identified that most users do not share content (free-ride) and that a large percentage of nodes had poor network connectivity. More recently, the problem of free-riding on Gnutella was revisited in [4]. The authors found that a greater number of users, as a percentage of the total users, are free-riding on Gnutella than in 2000. Queries forwarded to free-riding users will not yield any responses. Thus free-riding users degrade the high-level utility of the system and the low-level performance

A more closely related work to this paper is by Rasti et al. [9]. Their aim was to analyze the evolution of the v0.6 two-tier architecture with respect to the topological properties of the network. The authors measured an increase in the number of nodes in the network and showed that as nodes began to transition to the v0.6 protocol, the network became unbalanced with respect to the number of leaf and ultra-peers. They showed that modifications to major Gnutella client software and rapid adoption rate by users helped restore the overlay's desired properties. Our work is similar in that we analyze Gnutella's evolution from the v0.4 to the v0.6 architecture. However, our work focuses on the evolution of traffic characteristics and query performance. This allows for better understanding of the effects of architectural changes to the system. Further, understanding the evolution of the system's workload enables better decisions about future changes to the system and facilitates better modeling and simulation of large-scale P2P networks.

4 Results

4.1 Methodology

We captured traces for two weeks in May and June of 2003 as well as October of 2005, and June through September of 2006. We modified the source code of Phex [7], an open source Gnutella client written in Java to log traffic on the network. The client would connect to the network and log every incoming and outgoing

message that passed through it. It is important to note that our traffic capturing client does not actively probe the system; it only logs messages that it receives and forwards. Each log entry contained the following fields: timestamp, Gnutella message ID, message direction (incoming/outgoing), message type, TTL, hops taken, and the size of the message. In addition to logging incoming and outgoing messages, the client also logged queries and query responses in separate log files. For queries, we logged the query string. In the case of query responses, the log contained the Gnutella message ID of the original query, the peer ID where the matching objects are located and a list of the matching objects which includes the filename and the size of the file. The original protocol did not have separate classes of peers. As such, the traces from 2003 were collected with our client running as a regular peer. With the introduction of the v0.6 protocol, peers were classified as either leaf or ultra-peers. Our traces from 2005 and 2006 were captured with our client acting as an ultra-peer. The client was allowed to run 24 hours a day and logs were partitioned into two hour interval for ease of processing. These traces give insight as to the evolution of the Gnutella traffic over the last several years. We use the data from these traces to evaluate our design choices. In the interest of space, we present only results from one 24 hour trace from a typical weekday. The same day of the week was used (Thursday) for the traces of each year. Analysis of other days show similar results as those presented in this paper with weekends showing only 3% more traffic than weekdays.

4.2 Summary of Data

The average number of messages (incoming and outgoing) handled by the client was 2.5M in 2003 and 2.67M in 2006 per 2 hour interval. The average file size for a 2 hour trace was 292MB in 2003 and 253MB in 2006. This represents over 3GB of data per 24 hour period. In 2003, a 24 hour trace saw 2,784 different peers directly connected to it. In 2006, our client only saw 1,155 different peers directly connected to it in a 24 period. Although our 2006 traces saw fewer peers, other studies [8,9] show that the network has grown over time. The reduction in number of peers seen by our client can be attributed to a trend toward longer session times. In 2003, more than 95% of the sessions lasted less than 30 minutes. In 2006, more than 50% of the sessions lasted longer than 60 minutes. Longer session times imply slower churn rate for the connections at each node and thus our client would not see as many connections when the session times for peers is longer.

It is important to note that our data from 2005 shows characteristics that are consistent with the evolution of the protocol with respect to TTL and hops taken for messages. However, the bandwidth data is somewhat skewed. Investigation of this data revealed that although the client attempted to act as an ultra-peer, it was not able to maintain many connections (< 10). In contrast, the same client in 2006 was able to maintain over 30 connections consistently. Our traces from a client running as a leaf node show different behavior with a significant reduction in bandwidth for leaf nodes in 2006.

Table 1. Mean TTL left and Hops Taken for different message types in a 24 hour period from 2003, 2005, and 2006 data

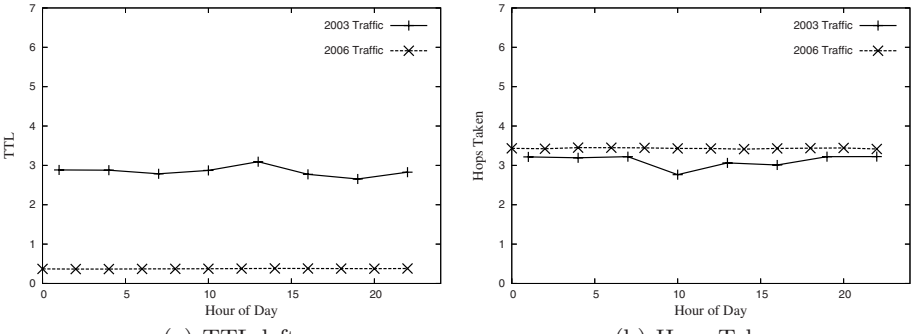
	All Messages		Control Messages		Query Messages	
	TTL Left	Hops Taken	TTL Left	Hops Taken	TTL Left	Hops Taken
2003	2.39	4.46	2.84	3.11	2.36	4.57
2005	2.95	3.41	3.17	3.76	2.52	2.71
2006	0.52	3.34	0.37	3.42	2.27	2.47

4.3 Message TTL Analysis

The TTL and the number of hops taken for each give an indication of how far messages travel in the network. Table 1 shows the mean TTL and number of hops taken for different classes of messages in a 24 hour trace for 2003, 2005 and 2006 traffic. In 2003, messages travel 6.85 hops into the network. Messages required 4.46 hops to reach our logging client and arrived with a mean TTL left of 2.39. Similarly, in 2005, messages travel 6.31 hops into the network requiring 3.41 hops to reach our client and arriving with a TTL of 2.95. In contrast, traffic from 2006 travels 3.86 hops into the network requiring 3.34 hops to reach our client and arriving with a mean TTL left of 0.52. This change can be attributed to the v0.6 protocol. The new protocol typically restricts the initial TTL of the message to between 4 and 5 hops to reduce the burden on the network due flooding from ultra-peers with many neighbors. A closer inspection of the TTL left and hops taken for the different types of messages reveals a change in the traffic characteristics of the different message types. In 2003, control messages (Ping and Pong) traveled further into the network and had to be propagated to many peers. On average, 2003 control traffic had a mean of 3.1 hops taken and a mean TTL left of 2.8. In contrast, 2006 control traffic had a mean of 3.4 hops taken and a mean TTL left of 0.37. Although messages arrive at a node from similar distances in 2003 and 2006, they are not expected to be propagated as far in 2006 as they were in 2003. Unlike control traffic, query traffic in 2006 is expected to be propagated further at each node. In addition, the TTL left for query traffic has remained fairly stable from 2003 to 2006: 2.36 in 2003 and 2.26 in 2006.

We should note the trend between 2003, 2005 and 2006. In 2005, control messages traveled deep into the network (6.93 hops) compared to 2006 (3.79 hops). In contrast, query messages traveled 6.93 hops in 2003, 5.23 hops in 2005 and 4.68 in 2006. This shows that query messages did not travel as deep into the network in 2005 as they did in 2003. This can be attributed to the transitioning to the current state of the v0.6 protocol. The protocol specification only sets guidelines as to what values to set for the initial TTL of a message. In 2005, vendors of Gnutella clients had already begun to set lower TTL values for query messages, but had continued to use a TTL of 7 for control messages. In 2006, these vendors transitioned to lower TTL values for all messages. In Figures 1 and 2 we show the hourly mean TTL and hops taken for control and query

traffic respectively. The traffic represents data from 24 hour traces from 2003 and 2006. We note that the TTL left and hops taken for each message remain relatively constant throughout the day even though bandwidth consumption and the number of nodes in the network fluctuate throughout the day. This observation can be used to develop traffic models when simulating a P2P file-sharing workload.



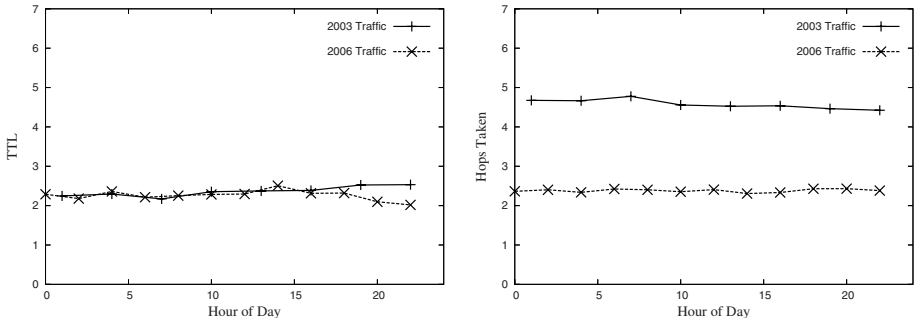
(a) TTL left (b) Hops Taken

Fig. 1. Mean hourly TTL and hops taken for incoming control messages in a 24 hour period for 2003 and 2006 traffic

4.4 Bandwidth Analysis

In this section we analyze the traces to identify changes in bandwidth consumption. We track the bandwidth utilization for control (Ping and Pong), query (queries and query hits), and route table update messages. We limit the control messages only to the basic Ping and Pong messages as they make up the overwhelming majority of non-query messages. Additionally, by tracking route table updates separate from the basic control messages, we can determine how the behavior of the network has evolved with respect to the use of each message type. Note that we do not show bandwidth consumed by peers for downloading objects.

We examined bandwidth utilization by tracking the bandwidth requirements for each message type. Figure 3 shows the bandwidth utilization for each message type in a 24 hour period for 2003 and 2006 traffic respectively. In 2003, query traffic dominates the bandwidth utilization. Further, the amount of query traffic varies at different times of the day while control traffic remains relatively stable throughout the entire day. The 2006 traffic exhibits different properties. First, the query traffic represents significantly less bandwidth than in 2003, although out going query traffic is still the dominant consumer of bandwidth. Control traffic, both incoming and outgoing, plays a larger role under the current v0.6 protocol in 2006 than it did in 2003, as does route table update traffic. The changes in bandwidth throughout the day are most pronounced in incoming and outgoing

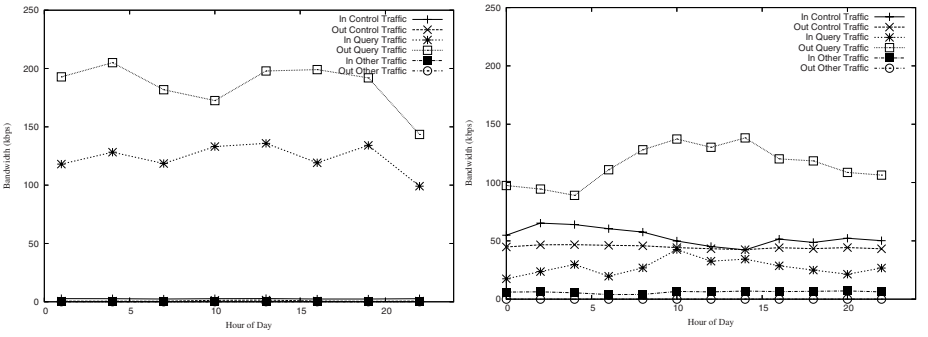


(a) TTL left (b) Hops Taken

Fig. 2. Mean hourly TTL and hops taken for incoming query messages in a 24 hour period for 2003 and 2006 traffic

query traffic while control traffic remains relatively constant throughout the day. The increase in route table update traffic can be attributed to the fact that in 2003, very few clients were using the new v0.6 protocol. As more clients moved to the new protocol, we see a rise in the number of route table update messages sent in the system.

The evolution of the Gnutella protocol and network results in traffic characteristics that are different in 2006 than they were in 2003. First, in 2003, control and route table updates represent a small percentage of the total bandwidth utilization. In 2006, however, control traffic is significantly more prominent with respect to the percentage of bandwidth utilization. Additionally, route table updates become a non-trivial message category in 2006 whereas these messages were virtually non-existent in 2003. Finally, the characteristics of query traffic from 2003 to 2006 change dramatically. In 2003, query traffic (both incoming and outgoing) dominates the bandwidth utilization. In 2006, query traffic still consumes a large percentage of the bandwidth. However, only outgoing query traffic consumes a large amount of bandwidth. Incoming query traffic uses less bandwidth than control traffic in 2006. This is a result of the evolution of the protocol specification. The v0.6 protocol establishes ultra-peers as peers with many neighbors. Ultra-peers can collect information about files shared at their leaf nodes and use this information to respond to queries. We saw earlier that message TTL are decreased in the v0.6 protocol, so fewer messages are sent in the network. Additionally, the v0.6 query routing protocol attempts to minimize the number of query messages that are sent. Therefore, a node under the current v0.6 protocol will receive fewer incoming query messages than it would have in 2003. Because ultra-peers have many connections, propagating queries to each neighbor results in outgoing query traffic utilizing a large percentage of the total bandwidth. Although the outgoing query bandwidth is large in absolute terms, there is an improvement relative to 2003 since the outgoing query bandwidth



(a) 2003 Traffic (b) 2006 Traffic

Fig. 3. Hourly bandwidth for different message types in a 24 hour period for 2003 and 2006 traffic. Note that the curves in Figure 3(a) do not wrap around to the same value at the end of a 24 hour period. This was due to our gathering procedure in 2003 that required the client to shut down after each 2 hour trace for processing. The client did not get restarted again for another hour in order to accommodate processing. This problem was fixed in our later data capturing process.

was reduced from 200 kbps to 100 kbps when the node is an ultra-peer with 30 connections. Leaf peers benefit from this situation as they have fewer connections and receive very few incoming queries.

4.5 Query Traffic Analysis

In the previous section we showed the evolution of bandwidth utilization for the different message types from 2003 to 2006. In this section, we will show the evolution of query traffic. We are interested in examining the relationship between the number of query messages sent and received and the success rate. Table 2 shows a summary of the query traffic measurements for traces from 2003 and 2006 traffic. The values in the table are for a typical 24 hour interval. In 2003, query traffic constituted a large component of the bandwidth used. We see in the table that in 2003, a peer would receive over 5M query messages in a 24 hour interval, or approximately 60 queries per second. In 2006, this number is significantly reduced to 280K queries in a 24 hour interval, or about 3 queries per second. With a mean query message size of 73.25 bytes for 2003 traffic, 60 queries per second corresponds to an incoming data rate of 4.39 KB/s or 35.2 kbps. Such a data rate would overwhelm most home dial-up connections. The current state of the network has reduced the number of queries received at each node at the expense of requiring a large number of connections for ultra-peer nodes. The large number of connections, in turn, results in a large outgoing bandwidth utilization. On a per query basis, 2003 traffic generated 0.285 KB per query, while 2006 traffic generated 4.026 KB per query. In 2003, each query was propagated to less than 4 peers on average while in 2006, each query was

Table 2. Query traffic summary for a 24 hour trace from 2003 , 2005 and 2006

	2003	2005	2006
Queries Received	5,261,064	614,365	279,235
Mean Queries per second	60.89	7.11	3.23
Query Message Size (including header)	73.25	69.04	105.61
Successful Queries	184,140	63,609	19,443
Success Rate	3.5%	10.3%	6.9%
Mean Queue Time (successful queries)	62.724 s	86.199s	5.435 s
Mean Outgoing Messages per Query	3.59	9.81	38.439
Mean Outgoing Bytes per Query	0.285 kB	0.677KB	4.026 kB
Mean Outgoing Query Bandwidth	138.82 kbps	38.5 Kbps	104.03 kbps

propagated to a mean of 38 peers. Note that in 2005, the bandwidth utilization is much lower compared to both 2003 and 2006. This is because our client was not able to operate as an ultra-peer and thus only was able to maintain less than 10 connections. However, other metrics such as incoming query rate for 2005 is consistent with the trend from 2003 to 2006.

Next we investigate the ability of the network to successfully resolve queries. We see that from 2003 to 2006, the success rate almost doubles from 3.5% in 2003 to 6.9% in 2006. Nevertheless, the success rate is still remarkably low. In 2006, a success rate of 6.9% implies that 93.1% of queries that reach a node will not be resolved after the node propagates the query. This means that each node is utilizing bandwidth to process queries but that effort is wasted on over 90% of those queries. In a 24 hour interval, a node receives approximately 280K queries with an average size of 105 bytes. Each of these queries is propagated to approximately 38 neighbors resulting in 1.14 GB of outgoing data, of which 93.1%, or 1.03 GB, are wasted since the success rate is only 6.9%.

5 Limitations

The study described in this paper was conducted using a single ultrapeer on our university's campus. As such, it may not be representative of the global behavior of Gnutella. We had performed measurements from a peer on a broadband network using a different service provider and observed similar results to as was reported in this paper. Also, the broadband studies were conducted in 2006; we do not have any measurement from 2003 or 2005 for this analysis. Hence, we cannot ascertain the consistency of the results among broadband users.

6 Conclusion

We presented the results of our measurement and analysis of the Gnutella file-sharing network. We showed that although the Gnutella architecture changed from 2003 to 2006 to help alleviate query bandwidth utilization, the success

rate of queries has not shown significant improvements. Additionally, the bandwidth utilization problem is alleviated at low-capacity peers (leaf peers), but high capacity peers (ultra peers) experience an increase in query bandwidth utilization. The increase in bandwidth occurs due to the large number of neighbors connected to the ultra peers. These findings indicate that despite the efforts to improve the performance of Gnutella, search performance is limited by the design of the protocol. We are investigating an approach that exploits the underlying characteristics of the queries and the distribution of objects in the system in order to improve search performance.

References

1. Gnutella protocol v0.4. http://dss.clip2.com/GnutellaProtocol04.pdf.
2. Gnutella protocol v0.6. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.
3. Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329. ACM Press, 2003.
4. Daniel Hughes, Geoff Coulson, and James Walkerdine. Free riding on gnutella revisited: The bell tolls? *IEEE Distributed Systems Online*, 6(6), June 2005.
5. Kazaa media desktop. http://www.kazaa.com/us/index.htm.
6. Overnet. http://www.overnet.org/.
7. The phex gnutella client. http://phex.kouk.de.
8. Yi Qiao and Fabin E. Bustamante. Structured and unstructured overlays under the microscope - a measurement-based view of two p2p systems that people use. In *Proceedings of the USENIX Annual Technical Conference*, 2006.
9. Amir H. Rasti, Daniel Stutzbach, and Reza Rejaie. On the long-term evolution of the two-tier gnutella overlay. In *IEEE Golbal Internet*, 2006.
10. M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.
11. Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.