Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella

Evangelos P. Markatos
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH)
P.O.Box 1385, Heraklio, Crete, GR-711-10 GREECE
http://archvlsi.ics.forth.gr markatos@csi.forth.gr
appears in the 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid 2002

Abstract

Peer-to-peer computing and networking, an emerging model of communication and computation, has recently started to gain significant acceptance. This model not only enables clients to take a more active role in the information dissemination process, but also may significantly increase the performance and reliability of the overall system, by eliminating the traditional notion of the "server" which could be a single point of failure, and a potential bottleneck.

Although peer-to-peer systems enjoy significant and continually increasing popularity, we still do not have a clear understanding of the magnitude, the traffic patterns, and the potential performance bottlenecks of the recent peer-to-peer networks.

In this paper we study the traffic patterns of Gnutella, a popular large-scale peer-to-peer system, and show that traffic patterns are very bursty even over several time scales. We especially focus on the types of the queries submitted by Gnutella peers, and their associated replies. We show that the queries submitted exhibit significant amounts of locality, that is, queries tend to be frequently and repeatedly submitted. To capitalize on this locality, we propose simple Gnutella caching mechanisms that cache query responses. Using trace-driven simulation we evaluate the effectiveness of Gnutella caching and show that it improves performance by as much as a factor of two.

1 Introduction

Recently, a new model of communication and computation, called peer-to-peer networking, has started to gain significant acceptance [3]. Contrary to the traditional client-server model, peer-to-peer computing enables all clients to act as servers and all servers to act as clients. In this way, clients not only take a more active role in the information dissemination process, but also may significantly increase the performance and reliability of the overall system, by eliminating the traditional notion of the "server" which could be a single point of failure, and a bottleneck in the overall system.

The first and most widely-known peer-to-peer system, Napster, [1] is a file sharing utility that has enabled hundreds of thousands of users to efficiently share files, including mp3-encoded songs, over the Internet. Capitalizing on the success of Napster, several other peer-to-peer file sharing systems have been recently developed. These include Gnutella, KaZaA, AudioGalaxy, etc. Although the technical details of these systems vary significantly, they all share the peer-to-peer philosophy by enabling all peers to store and deliver content to other peers. Besides file sharing, peer-to-peer systems have also been used for the efficient execution of highly parallel and distributed applications by capitalizing on the availability of idle cycles in home computers. Such applications range from systems that search for extra-terrestrial intelligence [7], to systems that seek a cure for AIDS [5].

Although peer-to-peer systems appeared only recently, their popularity has increased rapidly in the last couple of years. For example, network traffic measurements at the University of Wisconsin suggest that in the period of April 2000, Napster-related traffic represented the 23% of their total network traffic, while at the same time web-related traffic accounted for only 20% [12, 13]. Although Napster traffic has been reduced recently, the percentage of peer-to-peer traffic (in total) has actually increased. For example, recent measurement from the University of Wisconsin suggest that, in October 2001, peer-to-peer traffic reached more than 30% of the total traffic[2], while at the same time, web-related traf-

[1] http://www.napster.com
[2] http://wwwstats.net.wisc.edu/

fic was a little more than 19%.

Although these measurements suggest that the traffic demands of peer-to-peer systems represent a significant and continually increasing percentage of the overall network traffic, we still do not have a clear understanding of the magnitude, the traffic patterns, and the potential performance bottlenecks of such peer-to-peer networks.

In this paper we study the traffic patterns of Gnutella, a popular large-scale peer-to-peer system. We especially focus on the types of the queries submitted by Gnutella peers, and their associated replies. We identify the locality patterns that exist in Gnutella queries and propose simple, but effective caching mechanisms to exploit them.

More accurately, the contributions of this paper are:

- We installed three Gnutella clients, in two different continents and concurrently gathered traffic traces of Gnutella queries. We studied the traffic incurred by Gnutella query requests and query responses and found that this traffic is bursty and continues to remain bursty over several time scales.

- We studied the Gnutella query traces and showed that they show a significant amount of temporal locality, that is, several Gnutella queries were submitted more than once throughout the duration of our tracing study. Actually, the average Gnutella query was submitted between 2.5 and 5 times.

- We proposed and studied a simple caching policy that caches Gnutella query results for a limited amount of time. Our trace-driven simulation results show that even such a simple policy can significantly reduce the network traffic generated by Gnutella client, by as much as factor of two, while needing a very small amount of main memory.

The rest of the paper is organized as follows: Section 2 presents the methodology used to gather Gnutella traces. Section 3 compares our approach with previous work and places our paper in the appropriate context. Section 4 presents the traffic characteristics of the Gnutella network. Section 5 presents and evaluates our caching approach. Finally section 6 concludes the paper.

2 Methodology

2.1 The Gnutella Architecture

Gnutella is an overlay network superimposed on top of the Internet. Gnutella peers connect with their neighbors using point-to-point connections. In order to locate a file, a peer sends a query request to all its neighbors, which forward the query to their neighbors (as shown in figure 1 (a)), and so

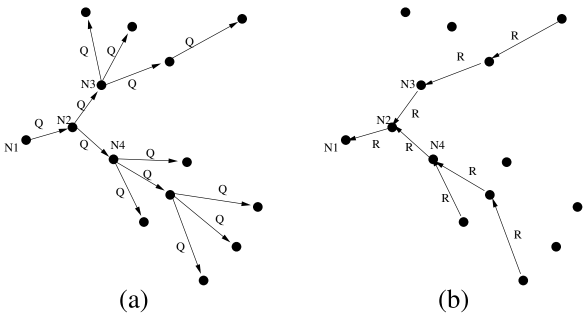

Figure 1. Query Requests (a) and Responses (b).

on. When a peer receives a query request, it searches its local files for a match to the query and returns a query response containing the matches it found (as shown in figure 1 (b)). Query responses follow exactly the reverse path of query requests. To avoid query requests from flooding the network, each query has a TTL (time to live) field, which is usually initialized at 7. When a node receives a query request with a positive TTL, it decrements the TTL field before forwarding the query. Queries received with a TTL equal to one are not forwarded. Therefore, queries may not travel more than seven hops in the network, and thus, the Gnutella network is free from never-ending queries. Although a Gnutella query can not circle the Gnutella network forever, it is possible to visit the same node more than once within the seven hops of its "life". To make sure that each node does not serve the same query twice, each query request is identified by an (almost) unique identification called `guid`. When a node receives a query with a guid it has encountered in the past, it simply drops the query.

2.2 Network Monitors

In order to monitor the Gnutella network and gather trace information, we made modifications to `gnut`, a UNIX-based open-source client for Gnutella [3].

We installed three `gnut` tracing probes: one in Greece (Crete), one in Norway (Bergen), and one in USA (Rochester, NY). We started the three tracing tools simultaneously on Thursday Oct. 4, 10 am EST. The tracing lasted for about one hour. Each probe recorded the queries and replies received. From the received queries, each probe removed the duplicates, that is, the queries with the same `guid` that have already been seen and forwarded in the past.

3 Previous Work

The study of large-scale peer-to-peer systems in general, and Gnutella in particular, is a rather new topic.

[3]http://www.gnutelliums.com/linux_unix/gnut/

Adar and Huberman studied the Gnutella traffic for a 24-hour period [1]. They found that close to 70% of the users shared no files, and that 50% of all responses were returned by only 1% of the hosts. Their findings were independently confirmed by Saroiu et al. who found that "there is significant heterogeneity and lack of cooperation across peers" participating in Gnutella [15]. Saroiu et al. findings suggests that a small percentage of the peers appeared to have "server-like" characteristics: they were well-connected in the network and they served a significant number of files. This disparity between the peers' characteristics may significantly limit the scalability, reliability and performance of the Gnutella network, and of peer-to-peer systems in general.

Anderson [2] observed the traffic of Gnutella for a 35-hour period and reported several results, including the distribution of TTL values, the distribution of Hops for Queries, the distribution of Hops for all Packets, etc.

Ripeanu *et al.*, studied the topology of the Gnutella network [14] over a period of several months, and found several interesting properties. Among them is that the Gnutella network topology does not match well the underlying Internet topology leading to the inefficient use of the network bandwidth.

Jovanovic [6] studied several Gnutella connection graphs and identified significant performance problems, including *short-circuiting*, an effect that limits the reachability of the nodes in a Gnutella network. Jovanovic's experiments suggest that, due to short-circuiting, a typical Gnutella peer reaches only about 50% of the peers that it could typically reach.

Sripanidkulchai [16] studied Gnutella traffic and, much like this paper, proposed the use of caching to improve performance. There exist however, several differences between our work and that of Sripanidkulchai. First, we collect Gnutella traces simultaneously from three different points on the Globe in order to make sure that our approach is not specific to one geographic region. Second, our definition of caching is fundamentally different from that in Sripanidkulchai . For example, Sripanidkulchai [16] caches query results independently from the node that made the query, which, in return, may result in delivering incomplete query responses. Furthermore, our performance metrics completely differ from that used in [16]. While Sripanidkulchai [16] uses the traditional performance metric of "hit rate", we show that in Gnutella query result caching, "hit rate" is not well defined and alternative performance metrics need to be considered.

Gnutella query caching, can be viewed as an extension of web document caching [17] and of search engine query caching [11, 9]. However, Gnutella query caching has significant differences from previous caching approaches including (i) different temporal locality characteristics, (ii) different performance benefits, and (iii) different staleness

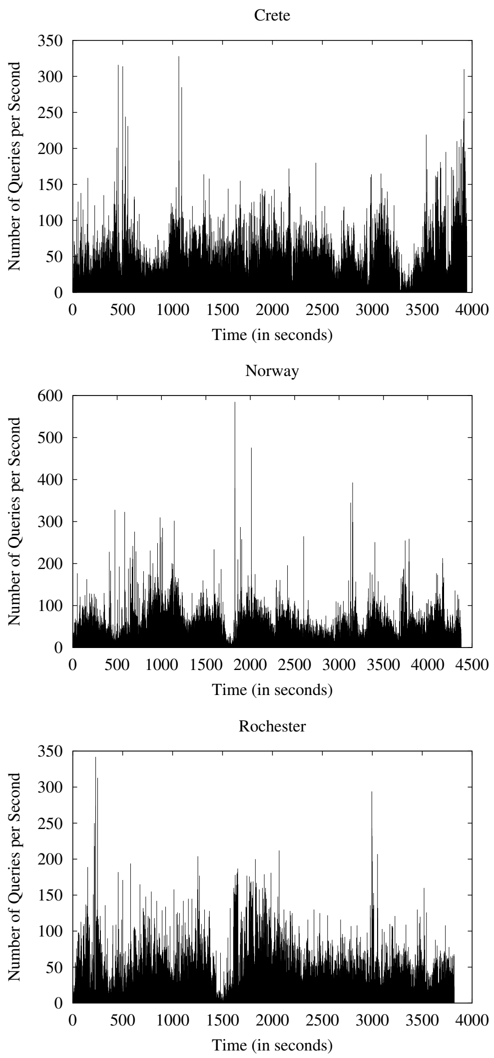

Figure 2. Queries per second.

properties. For example, a search engine's query result can safely be cached for several hours (if not for days). On the contrary, Gnutella query results may easily become stale within a few minutes.

4 Measurements

4.1 Gnutella Traffic Characteristics

4.1.1 Gnutella Query Requests

In our first experiment we report the number of the Gnutella query requests observed by each client. Table 1 shows the average number of queries received per second for each of the three clients. We immediately notice that all clients received a similar amount of query requests ranging between

Host	Query Requests/sec (avg.)
Rochester	45.9
Crete	47.9
Norway	52.3

Table 1. Queries per second - Overall average.

Host	Responses per sec (avg.)
Rochester	32.2
Crete	44.2
Norway	26.9

Table 2. Query Responses per second - Overall average.

46 and 52 requests per second. It is interesting to note that the geographic location of a client did not seem to have a direct effect on the number of queries it receives. For example, the US-based client (in Rochester) received less query requests per second than the client in Southern Europe (Crete). This is probably due to the time difference between Europe and the States, and/or because the topology of the Gnutella network does not necessarily follow that of the underlying geographical network, and therefore, clients that are geographically away from the United States may still receive a large amount of traffic. Figure 2 shows the actual number of queries received per second by each one of our three clients as a function of time. We immediately see that the load of each client varies very rapidly with time. For example, the query requests per second received by the client in Norway were between 1 and 585. That is, the client's load varied by as much as three orders of magnitude. Similarly, the load of the other clients as well varied 2-3 orders of magnitude.

To see if this burstiness of traffic holds over several time scales we plot the number of queries submitted per time interval (as a function of the time interval). [4] We use intervals of one second, 10 seconds, 1 minute, and 5 minutes. Figure 3 shows that for small intervals, (one and ten seconds) the traffic is very bursty and varies by as much as 2-3 orders of magnitude. The traffic remains bursty even for larger intervals (one minute and five minutes long), although the burstiness does not exceed one order of magnitude.

4.1.2 Gnutella Query Responses

In our next experiment we investigate what are the traffic patterns that result from responses to Gnutella queries. Table 2 shows the average number of responses per second that were observed by each of our clients. We see that the client in Crete received more than 44 responses per second, the client in Rochester received 32 responses per second, and the client in Norway received 27 responses per second. Figure 4 plots the actual number of responses seen per second by each one of the clients. We immediately see that the traffic due to responses is very bursty. For example, there were times where the clients received more than 1,000 query responses within one second. These were responses to queries that matched a lot of files. For example, one of the most popular queries we encountered in our measurements was the query "game" that matched more than 7,000 files, including several computer games, as well as songs and images that happened to have the work "game" in their titles. [5]

[4] In the interest of space we show the results only for the client in Norway. The results for the other clients are similar.

probe location	TTL > 0	TTL > 1
Crete	2.9%	10.33%
Rochester	3.2%	11.51%
Norway	3.4%	12.02%

Table 3. Percentage of Queries that had at least one hit.

Although some queries produce a large number of responses, most queries produce no responses at all. Table 3 (second column labeled "TTL>0") shows the percentage of queries that produced at least one response (at least one hit). We see that this percentage is between 2.9% and 3.4% for all our clients. These low percentages is probably due to the fact that the clients we installed for measuring Gnutella traffic share no files, and therefore could not generate a reply to any query. Thus, all query requests that terminate in our clients (i.e. have a TTL equal to one), are not forwarded to their neighbors and do not generate any responses. To factor out the effects of our clients, we measured the number of responses as a percentage of the queries that had a TTL greater than one. Queries with TTL greater than one were not terminated by our clients - they were forwarded to other clients. Therefore, we removed the queries with TTL equal to one from our calculations and measured the responses to queries that had TTL greater than one. The resulting percentage is shown in the third column of table 3. We see that only 10%-12% from the queries (with TTL greater than one) had any responses. Even though this percentage is higher than the one measured over all queries, it is still very low: roughly speaking, nine out of ten queries generate no response.

To see if the burstiness of Query responses holds over several time scales, we plot the number of query responses

[5] Interestingly enough, the query matched even some misspelled song titles, like "Theme-Games Bond 007-Golden Eye.mp3".

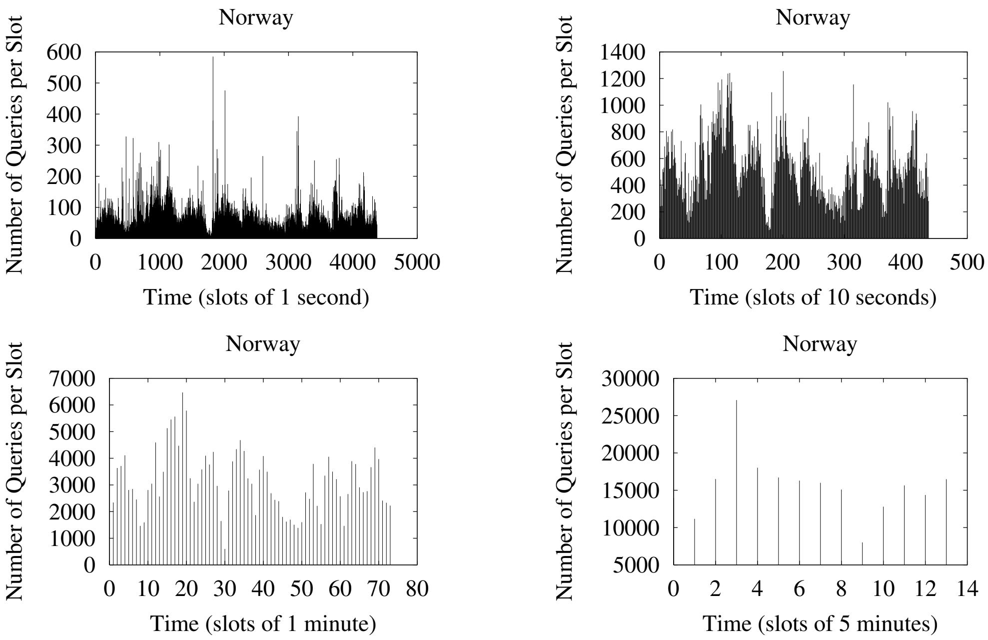

Figure 3. **Query Requests per time interval.**

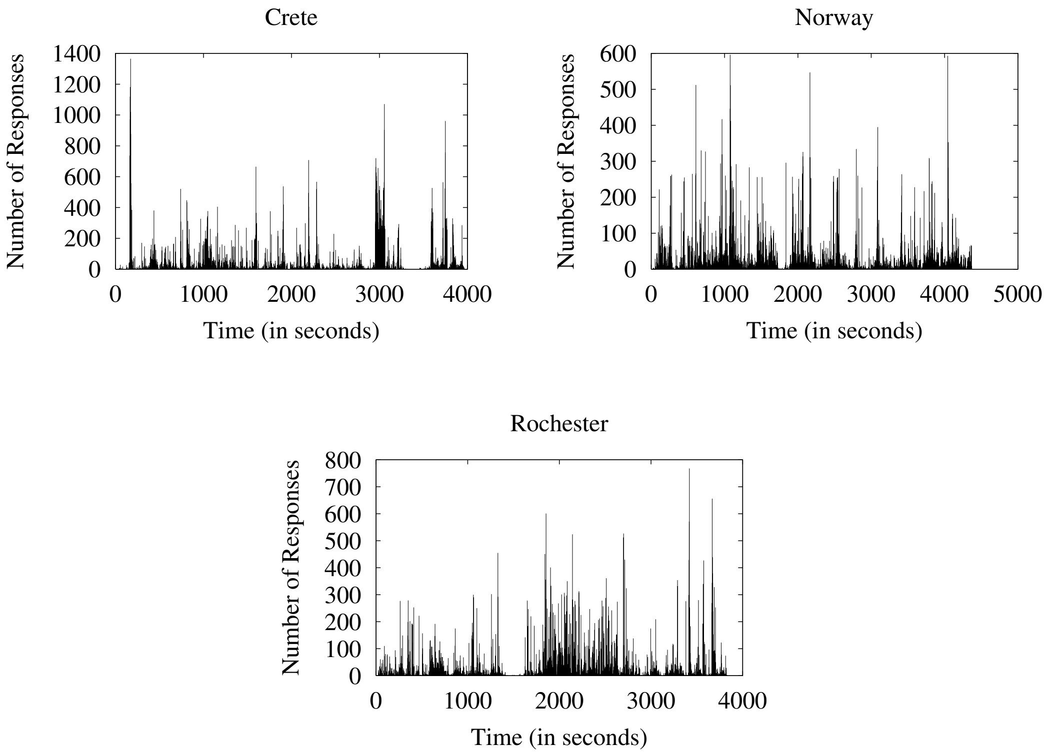

Figure 4. **Query Responses per second.**

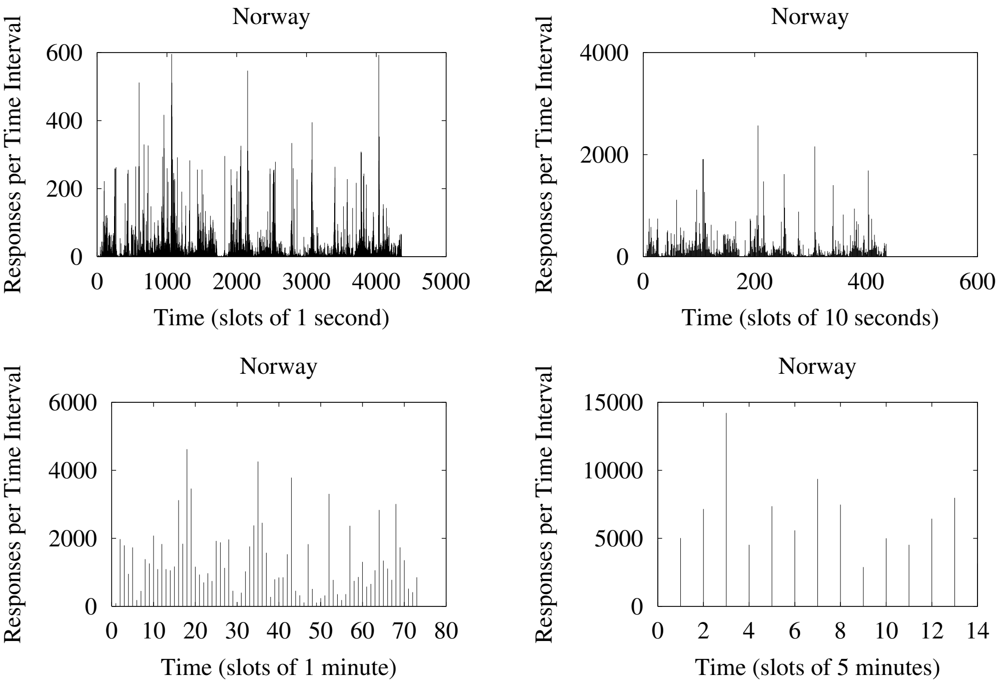

Figure 5. Query Responses per time interval.

received per time interval (as a function of the time interval) in figure 5. [6] We use intervals of one second, 10 seconds, 1 minute, and 5 minutes. We see that for small intervals, (one and ten seconds) the traffic is very bursty and varies by as much as 2-3 orders of magnitude. However, we see that the traffic remains bursty even for larger intervals (one-minute and five-minute intervals). Similar observations about the burstiness of Internet traffic have been reported before for the Internet and the web [4, 8].

4.1.3 TTLs: life of queries

In our next experiment we measure the distribution of TTLs of the various queries. Figure 6 plots the number of queries that had a TTL between one and seven. [7] We see that the number of queries decreases exponentially with increasing TTL. For example, the client in Crete sees 115 queries with TTL equal to seven, 459 queries (4 times more) with TTL equal to six, 915 queries (8 times more) with TTL equal to five, 3460 queries (30 times more) with TTL equal to four, 10129 queries (88 times more) with TTL equal to three, 36087 queries (313 times more) with TTL equal to two, and 137927 queries (1200 times more) with TTL equal to one. We see that in all cases the number of queries with TTL equal i is 2-4 times larger than the number of queries with TTL equal to $i+1$. This "exponential" behavior is probably due to the broadcast approach followed by the Gnutella protocol: each Gnutella peer forwards each query to its neighbors, which in turn forward the query to their neighbors, and so on. Therefore, the number of Gnutella query messages increases (roughly) exponentially as the query is propagated in the network. Recall, that each time a query request is forwarded, its TTL is decremented by one. Therefore, query requests with small TTLs are exponentially more than query requests with large TTLs. To illustrate this point, figure 7 shows an example of a Gnutella network and the distribution of the TTLs. We see that the root node receives a query with TTL equal to 5, which by forwarding the query to its neighbors generates two queries with TTL equal to 4. These neighbors forward the query one more hop generating 4 queries with TTL equal to 3, and eventually 8 queries with TTL equal to 2. We see therefore that as a query propagates in the Gnutella network, it generates an exponential number of queries which have a TTL that decreases linearly with the number of hops.

4.2 Locality

Our experiments so far have investigated the traffic patterns of the Gnutella query requests and their associated

[6] Due to space limitations we again show the results only for the client in Norway. The results for the other clients are similar.

[7] Queries with TTLs lower than one are probably the result of erroneous clients. Queries with TTLs larger than 7 will generate excessive amounts of traffic and are usually not forwarded by Gnutella clients.

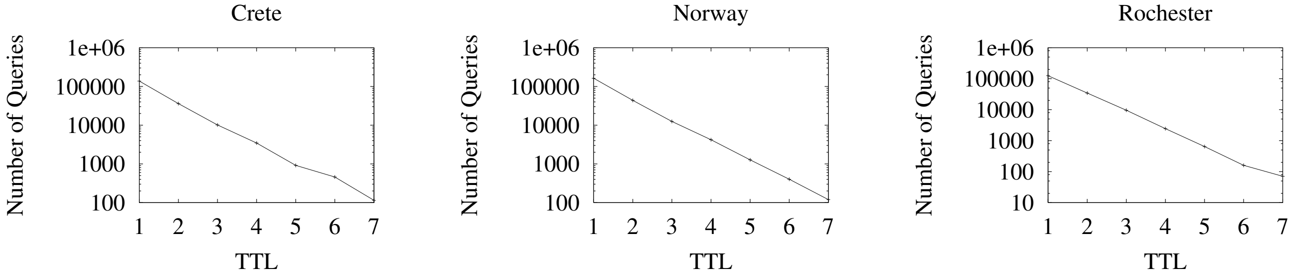

Figure 6. TTL distribution (for TTLs less than 8).

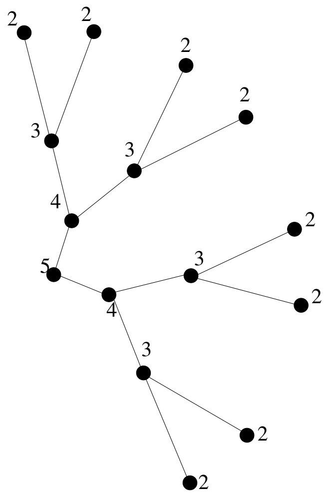

Figure 7. Distribution of TTLs in a Gnutella Network.

responses. We have already established that both query requests and query responses exhibit a very high degree of burstiness. We will now turn our attention into any locality patterns that may exist in Gnutella queries.

Table 4 shows the average number of times each query has been observed by our probes. The second column (labeled "TTL > 0") shows the average for all queries, and the third column shows the average for queries that have TTL greater than one. The latter are queries that are forwarded to other clients. We see that the average number of submissions of each query is between 4.5 and 5 (over all queries), and between 2.6 and 3 (for queries with TTL greater than one). Figure 8 shows the percentage of queries that were submitted only once, the percentage of queries submitted twice, and so on. We see that in all cases, 60% of the queries were submitted only once, and therefore about 40% of the queries were submitted more than once. Actually, about 10% of

probe location	TTL > 0	TTL > 1
Crete	4.5	2.74
Rochester	4.5	2.57
Norway	4.9	2.98

Table 4. Average number of times each query is submitted.

the queries were submitted twice, 5% of the queries were submitted three times, and so on. It is interesting to see that these percentages are almost identical for all clients.

5 Query Caching

Table 4 has already established that Gnutella query requests exhibit a significant amount of locality. In our next experiments we explore whether this locality can be exploited in order to reduce the overall network traffic. Locality is usually exploited (among other ways) by caching of the frequently accessed data. For example, web-caching and content delivery systems store copies of frequently-accessed data in proxies (caches) located close to the clients that request them [17].

However, caching query results in peer-to-peer systems, like Gnutella, is significantly different from caching (query results or other documents) in web caching systems. The main difference between Gnutella caching and web caching is that in traditional web caching, the "content", that is cached by proxies, is provided by well-defined web servers. On the contrary, in Gnutella, the "content" (which is actually the sum of the responses to a query request) is not provided by any well-defined single server, but is computed by composing the results of the content provided by several peers. Therefore, besides the peer that originally issued the query, no other peer has complete knowledge of the "content" of a query's response.

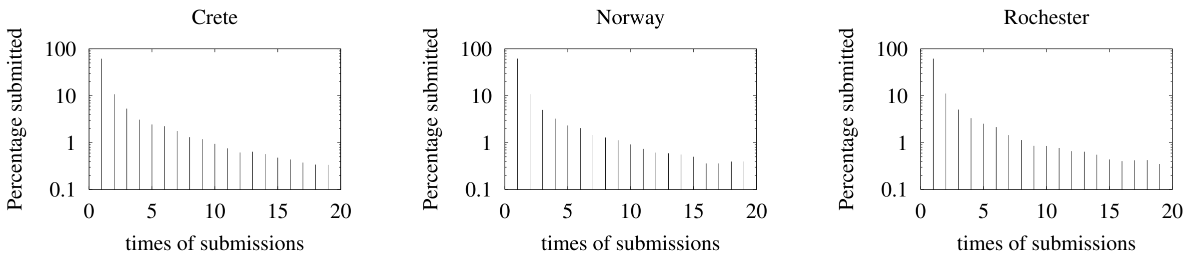

Figure 8. **Query popularity.**

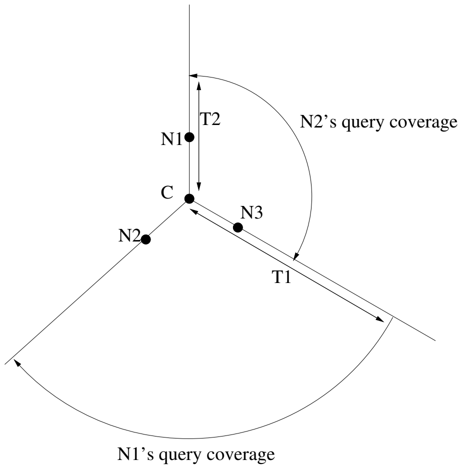

Figure 9. Query Caching and Coverage of Responses in Gnutella.

Let us illustrate the difference between web caching and peer-to-peer caching by an example shown in Figure 9: suppose that a Gnutella peer C receives a query request (e.g. "cnn") from its neighbor N_1 with TTL equal to T_1. C forwards the query to neighbors N_2 and N_3, and receives the query's responses which it later forwards to N_1. The responses that C has collected represent all the files that matched "cnn", and are reachable within T_1 hops from C through its neighbors N_2 and N_3. Suppose now, that at some later point in time, C receives the same query request ("cnn") from neighbor N_2 with TTL equal to T_2. In this case, C can not just give to N_2 the results of N_1's query request, because they do not include the files that match "cnn" that are reachable through N_1. Furthermore, since the TTL's of the two queries are different, the responses to each query correspond to a different coverage of the Gnutella network. Therefore, C can not just use the response of the N_1's query to satisfy N_2's query, even though the two query requests textually match each other. Thus, although a Gnutella peer may see a query request several times, these requests do not necessarily produce the same results, especially if they originate from different neighbors and have different TTLs. Therefore, Gnutella caching systems need to take into account not only the text of the query request but also the query's TTL, the neighbor that issued the request, and in general, all the factors that define the coverage of a query's response.

Although Gnutella Caching is different from web caching, it can still be employed in order to improve performance. Our proposed approach to Gnutella caching is as follows: When a client C receives a query from neighbor N_2, its checks its cache to see if a query with the same text and the same TTL has been seen in the past. If such a query if found, and if this query has been sent in the past by neighbor N_2, C returns the responses that exist in its cache for this query. If, on the other hand, C finds such a query in its cache, but the (cached) query had been sent by neighbor N_1, C forwards the (new) query to N_1, receives the results, combines them with the locally cached results of the query, and forwards the combined result to N_2. [8]

5.1 Cache Design

5.1.1 Cache Size

Traditional caching systems reserve a predefined amount of space for their cache and store in it as much useful data as they can. To make the best possible use of their limited-size cache, some web caching systems, give preference to small documents in order to fill the cache with as many documents as possible [10]. However, we believe that Gnutella caching should behave differently from other forms of caching, mainly due to the fact the Gnutella query responses are time-sensitive. The Gnutella network is highly dynamic: Gnutella peers join and leave the network very frequently. Therefore, the responses to a given query

[8] Note that before C can forward the list with the combined results to $N2$, it must first remove the cached results received by $N2$ in the past.

request may become out-of-date even after a small time period. Thus, if query responses in Gnutella are cached, they should be kept in the cache for only a small amount of time, that ranges from several seconds to at most a few minutes. Therefore, contrary to traditional methods of caching, the objective of Gnutella caching is not to keep a limited-size cache as full as possible, but to keep the data in the cache for a time period that is long enough to improve performance, but also short enough to avoid sending stale responses.

5.1.2 Evaluating Gnutella Query Caching

Traditionally, the performance metric that has been used to evaluate caching approaches is the (cache) hit rate. The cache hit rate is the percentage of requests that were actually found in the cache over the measured period of time. As we have already explained, although some Gnutella queries may find their entire set of responses in the cache, other queries may find only a portion of their responses in the cache. Although the first type of queries can be clearly categorized as cache hits, the second type of queries can be treated neither as cache hits, nor as cache misses. Even though the queries of the second type can not be clearly characterized as hits or misses, they can definitely participate in a caching system and improve performance by retrieving a potentially large portion of their responses from the Gnutella cache. These queries can help to reduce the overall network traffic, since Gnutella cache systems forward them to only to one neighbor, instead of all the neighbors. Therefore, to evaluate the effectiveness of Gnutella query caching we will not measure the "hit rate", but we will measure the network traffic reduction that is achieved by caching, or more precisely, the percentage reduction in the number of the query requests sent out in the network.

5.2 Network Traffic Reduction

We have designed and build a trace-driven simulator that takes as input the produced Gnutella query traces, simulates the caching approach we proposed, and measures the performance improvement of the overall system.

Figure 10 plots the reduction of the query request packets sent out in the network as a function of the caching interval. We see that caching query results even for as low as one minute reduces query request packets by as much as 15%-20%. Caching for 5 minutes results in a 30% reduction. Caching for half-an-hour results in close to 50% network traffic reduction. Effectively, caching query results for 30 minutes reduces the number of query requests by half. Our results agree with the query popularity patterns observed in table 4. For example, table 4 suggests that each query (with TTL > 1) is submitted on the average close to 2.5 times. Therefore, an *ideal* caching algorithm (that would cache queries indefinitely, and that would not take into account the sender of the query) could achieve a "hit rate" and an associated query request packets reduction of about 1.5/2.5=60%. Our simulations suggest that by using our less-than-ideal caching approach and by caching Gnutella query response for half an hour results in a query request packets reduction close to 50%, which is close to the "ideal" 60%.

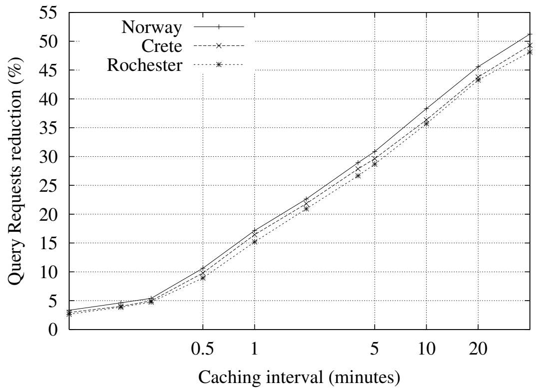

Figure 10. Reduction of Query Request packets sent out in the network.

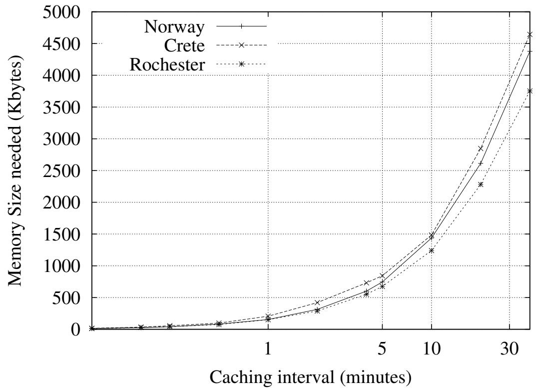

Figure 11. Cache Size needed.

5.3 Cache Size Requirements

Although caching reduces network traffic significantly, it requires that each peer contributes an amount of its local memory to store the cached query responses. If the size of this local memory (hereafter called the *cache size*) turns out to be large, it can be a limiting factor to the deployment of caching. This cache size includes the memory needed to store the query responses themselves, as well as all the metadata need to organize these responses into appropriate and efficient data structures. The reader may notice that this cache size may vary with time, because the proposed

Gnutella caching approach keeps the responses of only the last given time interval, and as shown in figure 4 the number of these responses may vary. Therefore, during a busy time interval, the cache size will be larger than it would be during a low-traffic time interval. To smooth these fluctuations, our results report the average amount of memory needed for the duration of the simulation. Figure 11 plots the average cache size as a function of the caching interval. We immediately see that the memory demands are very low. For example, caching query responses (and their associated metadata) for one minute required only 200 KBytes. Caching query responses for as long as five minutes required no more than 1 MByte of memory. Even caching for as long as 20 minutes required no more than 3 MBytes of memory. Therefore, the memory needs of Gnutella caching can be considered rather small. Most (if not all) Gnutella peers can easily invest a few hundred KBytes (or at most a few MBytes) of their memory in order to improve performance, and reduce the overall network traffic.

6 Conclusions

In this paper we studied the traffic of Gnutella, a large peer-to-peer application. We installed three Gnutella clients in three countries in two different continents. We identified the locality patterns that exist in Gnutella query requests, and proposed a simple caching policy that caches query responses for a short amount of time. Using trace-driven simulation we studied the performance of this policy. Based on our measurements and experimental evaluation we conclude:

- *Peer-to-peer Systems like Gnutella have a very bursty traffic pattern.* Both query requests and responses have very bursty traffic patterns even when observed over several time scales.

- *Queries submitted to Gnutella show a significant amount of locality*. Our measurements suggest that the average query has been submitted 2-5 times within a one-hour period.

- *Caching Gnutella queries even for a small amount of time may result in significant performance improvements.* Our trace-driven simulations suggest that caching Gnutella query responses for several minutes may reduce the query requests sent out in the network by as much as a factor of two.

- *Caching Gnutella queries requires only a small amount of memory*. Our experiments suggest that Gnutella query caching in most cases required no more than 1-3 MBytes of memory.

Overall, we believe that peer-to-peer caching systems are beneficial today and will be increasingly important in the near future when an even larger number of will join peer-to-peer networks.

Acknowledgments

This paper was supported in part by IST project SCAMPI (IST-2001-32404) funded by the European Union. We thank the staff of the University of Rochester and the University of Bergen for their help in gathering the traces. We would like to thank Catherine Chronaki and Elias Athanasopolous for their constructive comments.

References

[1] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] K. Anderson. Analysis of the traffic on the gnutella network, 2001. http://www-cse.ucsd.edu/classes/wi01/cse222/projects/reports/p2p-2.pdf.

[3] D. Clark. Face-to-face with peer-to-peer networking. *Computer*, 34(1):18–21, Jan. 2001.

[4] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions. Networking*, 5(6):835–846, 1997.

[5] FightAIDS@home. http://www.fightaidsathome.org/.

[6] M. Jovanovic. Modeling lareg-scale peer-to-peer networks and a case study of gnutella. Master's thesis, University of Cincinnati, 2001.

[7] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. Seti@home-massively distributed computing for seti. *Computing in Science & Enginering*, 3(1):78–83, 2001.

[8] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.

[9] Q. Luo, , and J. F. Naughton. Form-based proxy caching for database-backed web sites. In *Proceedings of the VLDB 2001*, 2001.

[10] E. Markatos. Main memory caching of web documents. *Computer Networks and ISDN Systems*, 28(7-11):893–906, 1996.

[11] E. P. Markatos. On caching search engine query results. *Computer Communications*, 24(2001):137–143, 2001.

[12] D. Plonka. Uw-madison napster traffic measurement, 2000. http://net.doit.wisc.edu/data/Napster/.

[13] D. Plonka. An analysis of napster and other ip flow sizes. *Network Analysis Times*, April 2001. http://moat.nlanr.net/NATimes/april2001.pdf.

[14] 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.

[15] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN) 2002*, 2002.
[16] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scaling, 2001.
[17] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46, 1999.