On the Equivalence of Forward and Reverse Query Caching in Peer-to-Peer Overlay Networks

Ali Raza Butt[1], Nipoon Malhotra[1], Sunil Patro[2], and Y. Charlie Hu[1]

[1] Purdue University, West Lafayette IN 47907, USA,
{butta,nmalhot,ychu}@purdue.edu
[2] Microsoft Corporation, One Microsoft Way, Redmond WA 98052, USA,
patro@microsoft.com

Abstract. Peer-to-peer systems such as Gnutella and Kazaa are used by millions of people for sharing music and many other files over the Internet, and they account for a significant portion of the Internet traffic. The traffic in a peer-to-peer overlay network is different from that in WWW in that each peer is both a client and a server. This suggests that one can deploy a forward cache at the Internet gateway of a network to reduce the amount of queries going outside, or a revere cache at the gateway to reduce the amount of queries going inside, which in turn reduces the queries that are forwarded outside. In this paper, we study the effectiveness of forward and reverse caching at the gateway via analysis and experimental measurement. Our study shows that forward caching and reverse caching at the gateway are equally effective at reducing query and query reply traffic across the gateway.

1 Introduction

In the first six years of the Internet explosion, one type of dominating traffic over the Internet had been HTTP traffic generated from widespread accessing of the World Wide Web. Around the year 2000, a new paradigm for Internet applications emerged and has quickly prevailed. Today, the so-called peer-to-peer (p2p) systems and applications such as Gnutella and Kazaa are routinely used by millions of people for sharing music and other files over the Internet, and they account for a significant portion of the Internet traffic. For example, the continuous network traffic measurement at the University of Wisconsin (`http://wwwstats.net.wisc.edu`) shows that peer-to-peer traffic (Kazaa, Gnutella, and eDonkey) accounts for 25–30% of the total campus traffic (in and out) in August 2002, while at the same time, web-related traffic accounts for about 23% of the total incoming and 10% of the total outgoing traffic.

The traffic generated by a p2p network such as Gnutella falls under two categories: the protocol messages for maintaining the overlay network and for searching data files, and the actual data messages for downloading files. Since the actual data download is performed through direct connections between the

In *Proceedings of the 9th International Workshop on Web Content Caching and Distribution (WCW'04)*, Beijing, China, October 18-20, 2004.

source and destination servents via HTTP, such traffic can be cached using off-the-shelf web caching proxies. Thus the more interesting question is how to reduce the protocol messages which are typically propagated in the overlay network via controlled flooding.

Similar to accesses of web content, researchers have found that search messages (queries) in p2p networks such as Gnutella exhibit temporal locality. This suggests that caching can prove to be an effective technique in reducing network bandwidth consumed by these queries and the latency in retrieving query replies. While the locality in web accesses is determined solely by the URLs being accessed, the locality in the queries in p2p networks also takes into account the TTLs and the next hop nodes in the overlay, as these two factors also affect the set of the query replies that will be received.

Several query caching schemes have been developed and have confirmed the effectiveness of query caching [1–3]. In particular, a transparent caching scheme has been proposed in [3] where p2p caching proxies are attached to the gateway routers of organizations or ISPs, i.e., similar to how web caching proxies are typically deployed, with the goal of reducing p2p traffic in and out of the gateways. The gateway router is configured to redirect TCP traffic going outside and to well known p2p ports, e.g., 6346 for Gnutella, to the attached p2p caching proxies. We call this *forward caching* as the the cache acts on outgoing queries.

A fundamental difference between the traffic in p2p overlay networks and web traffic is that a peer in a p2p network is both a client and a server. This observation suggests that one can also deploy a *reverse caching* proxy at the gateway of an organization to exploit the locality in queries coming into that organization. Intuitively, compared to a forward proxy, a reverse proxy will see fewer distinct queries since there are fewer peers within an organization than their neighbors outside, and query locality distinguishes queries sent to different forwarders. Consequently, reverse caching is expected to achieve a higher cache hit ratio than forward caching due to fewer capacity misses, and thus is seemingly more effective than forward caching in reducing query traffic across the gateway.

In this paper, we study the relative effectiveness between transparent forward and reverse caching at the gateway focusing on Gnutella networks. We present a detailed analysis that shows reverse caching and forward caching are equally effective assuming the caches store the same number of query hits, disregarding how many queries they are for. We further confirm our analysis with experimental results collected from a testbed running eight Gnutella servents behind the caching proxies.

2 Preliminaries

2.1 The Gnutella Protocol

To set up the context for the analysis of forward and reverse caching, we briefly discuss Gnutella's node joining process, its implications on the topology of the part of Gnutella network inside an organization, and the query request and reply protocols. The details of the Gnutella protocol can be found in [4, 5].

Topology of Gnutella networks inside an organization The joining process of a typical Gnutella servent is as follows. When a servent wants to connect, it first looks up its *host cache* file to find addresses of Gnutella servents to connect to. The servent addresses are removed from the host cache after they are read. If the host cache is empty, it tries to connect to a well known Gnutella *host cache server* (also called PONG server) to receive PONG messages in response to its PING message. After receiving the PONG messages, the servent tries to establish connections with the servents whose addresses are obtained from the PONG messages.

A typical Gnutella servent S, after establishing a pre-specified number of Gnutella connections, periodically sends PING messages to monitor these connections. In response S receives a number of PONG messages[3], which are appended at the end of S's host cache file. In addition, when an existing connection with some servent S_1 is broken down, S_1's address information is saved and eventually will be added to S's host cache when it leaves the Gnutella network.

In summary, during the joining process of a typical Gnutella servent, the neighbors are chosen from the host cache whose content is fairly random. This suggests that it is unlikely servents from the same organization will become neighbors of each other, and consequently query messages will travel across the gateway of the organization.

Query and query replies In order to locate a file, a servent sends a query request to all its direct neighbors, which in turn forward the query to their neighbors, and the process repeats. Each Gnutella query is identified by a unique tuple of `(muid, query string, forwarder, neighbor, ttl, minimum speed)` values, where the query is forwarded from the `forwarder` servent to the `neighbor` servent, and the `minimum speed` value specifies the minimum speed that a servent should be able to connect at if replying to the query. When a servent receives a query request, it searches its local files for matches to the query and returns a query reply containing all the matches it finds. Query replies follow the reverse path of query requests to reach the servents that initiated the queries. The servents along the path do not cache the query replies.

To avoid flooding the network, each query contains a TTL field, which is usually initialized to a default value of 7. When a servent receives a query with a positive TTL, it decrements the TTL before forwarding the query to its neighbors. Queries received with TTL equal to 1 are not forwarded.

2.2 Transparent Query Caching

The natural way of performing transparent query caching in p2p overlay networks is similar to transparent web caching. A caching proxy is attached to the gateway router, the router is configured to redirect outgoing TCP traffic to certain designated ports (e.g., 6346 for Gnutella) to the attached proxy, and the proxy hijacks such connections going out of the gateway. The proxy can then

[3] In Gnutella version 0.6, a servent uses its PONG cache to generate PONG messages.

cache query replies from outside the gateway and use them in the future to reply to queries from inside. Since the cached query replies are used to reply to queries going outside the gateway, we call this *forward query caching.*

However, queries can also come inside the gateway along the hijacked outgoing connections at the proxy. This is due to a fundamental difference between caching in p2p and web caching, that is, a peer node inside an organization acts both as a client and a server, while nodes involved in web traffic inside an organization are only clients (i.e., browsers). Therefore, in principle, the caching proxy can also inspect the queries that come from outside the gateway on the hijacked connections and cache query replies received from the nodes inside. The proxy can then use the cached query replies to serve future queries coming from outside along the hijacked connections. We call this *reverse query caching.*

Hijacking incoming connections The above transparent caching scheme only hijacks outgoing connections from servents inside the router, i.e., Gnutella connections that are initiated by inside servents to port 6346 of outside servents. The incoming connections from servents outside the router to servents inside can also be hijacked by configuring the external interface of the router to redirect incoming traffic to an attached proxy, similar to how outgoing connections are hijacked.

We note that the difference between our definitions of forward and reverse caching are orthogonal to whether the hijacked connections are outgoing or incoming connections in the overlay, since the query traffic going through them is indifferent to who initiated the connections.

2.3 The Caching Algorithm

We summarize the caching algorithm used by the proxy previously described in [3]. First, all the PING/PONG messages initiated or forwarded by the servents, inside or outside, going across the gateway will be forwarded by the caching proxy. The caching proxy will not change the TTL, and thus the reachability of PING/PONG messages remains the same as before. Similarly, HTTP data download messages will be tunneled through unaffected.

Our caching algorithm caches query hits according to the tuple of `(query string, neighbor, ttl)` values of the query they correspond to. It uses two main data structures. First, any time the proxy tunnels a query to outside servents, it records the `muid`, the query string, and the TTL information in a Cache Miss Table (CMT). When a query hit is received from outside, its `muid` is checked against CMT to find the corresponding query string and TTL, which is used to index into the cache table. The cache table (CT) is the data structure for storing cached query hits. For each CT entry CT(i), the algorithm adds a vector field to remember the `muid` and `forwarder` of up to 10 most recent queries for which query hits are replied using that cache entry.

Every time a new query results in a cache hit with CT(i), i.e., with a matching tuple `(query string, neighbor, ttl)`, the `muid` of the new message is compared with those stored in CT(i). If the `muid` matches that of any of the

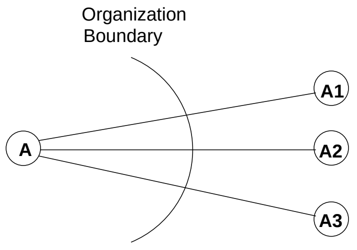


Fig. 1. Servent node *A* and its three neighbors outside the gateway.

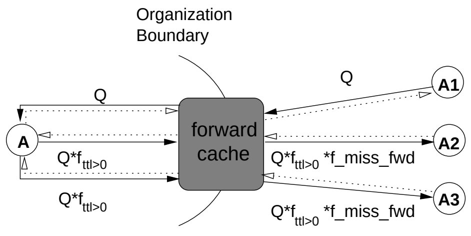
10 stored previous queries replied to by the proxy using CT(i), it suggests that the same query reached the proxy a short while ago, and the query is dropped. Otherwise, the proxy is seeing this query message with this `muid` for the first time, and hence the proxy replies from the cache. The `muid` of this new query is then stored in the vector field of the corresponding CT entry.

Speed rewriting Upon a cache miss, the proxy rewrites the minimum speed field of the query to zero before forwarding it to outside the gateway. As a result, it collects query hits with all possible speeds. For a subsequent query that matches all other parameters with the cached query hits, but specifies a non-zero minimum speed requirement, the proxy can always extract a subset of the cached query hits that satisfy the minimum speed requirement, without forwarding the query out of the gateway again.

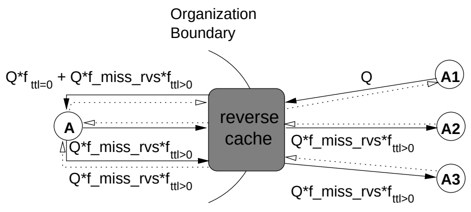
3 Analysis

In this section, we analytically compare the effectiveness of forward and reverse query caching at the gateway. Without loss of generality, we focus on a single Gnutella servent inside the gateway, which has C neighbors in the p2p overlay, all of which are outside the gateway (Section 2.1). The following comparison analysis between forward and reverse caching also holds true for multiple Gnutella servents inside the gateway if the multiple servents do not share any neighboring nodes outside the gateway. The disjoint neighborhood among multiple servents behind the gateway is expected because of the node join process as discussed in Section 2.1, and is also confirmed to be true in our experiments. Figure 1 shows servent A and its 3 neighbors in the p2p network outside the gateway.

We assume all servents in the p2p network initiate queries at about the same frequency. Since queries have a typical TTL of 7, each query is expected to reach a large number of servents in the p2p network. Conversely, compared to the queries generated by each servent, a much larger number of queries will reach and be forwarded by that servent. Thus we can ignore the queries generated by servent A in analyzing the query traffic related to it, e.g., in the scenario in Figure 1.



(a) Forward caching



(b) Reverse caching

Fig. 2. The query and query hit traffic on each hijacked neighbor connection of servent A triggered by Q queries initiated from servent $A1$, under forward and reverse caching at the gateway, respectively. Starting from $A1$, solid arrows show queries (1) received from outside, (2) sent to inside, (3) received from inside, and (4) sent to outside. Starting from $A2$ or $A3$, dashed arrows show query hits (5) received from outside, (6) sent to inside, (7) received from inside, and (8) sent to outside. $f\_ttl > 0$ denotes the fractions of queries with $ttl > 0$ after decrementing at A. f_miss_fwd and f_miss_rvs denote the cache miss ratio in forward and reverse caching, respectively.

We assume all connections between A and its neighbors are hijacked by the proxy. We analyze the amount of query and query traffic on each hijacked connection triggered by a set of Q queries initiated from or forwarded by a single neighbor – servent $A1$ – in the forward and reverse caching scenarios. The amount of queries sent on each connection are shown in Figure 2(a) and Figure 2(b) for forward and reverse caching, respectively.

The numbers of queries on each connection in forward caching shown in Figure 2(a) are explained as follows. First, the proxy receives all Q queries from servent $A1$. It then passes all of them through to servent A. Out of these queries, $Q \cdot f_{ttl>0}$ (after decrementing the TTL at A) will be forwarded to each of A's remaining $(C-1)$ neighbors in the p2p network, where $f_{ttl>0}$ is the fraction of queries with $ttl > 0$. These are the queries on which the proxy acts, i.e., by performing cache lookups. Finally, $Q \cdot f_{ttl>0} \cdot f_{miss\_fwd} \cdot (C-1)$ of them will be cache misses and forwarded to the $(C-1)$ neighbors outside the gateway, where f_{miss_fwd} is the query miss ratio in forward caching.

The numbers of queries on each connection in reverse caching shown in Figure 2(b) are explained as follows. Once again, the proxy receives all Q queries from servent $A1$. First, it forwards the ones with $ttl = 0$ (after decrementing the TTL) directly to A. The reason the proxy does not perform caching on queries with $ttl = 0$ is that such queries will not be forwarded further once reaching the servent inside the gateway, thus caching such queries will not reduce the outgoing query traffic. Furthermore, the proxy can avoid significant consumption of its cache capacity by such queries; around 60-70% queries received by any servent and thus going through the proxy have $ttl = 0$ (after decrementing the TTL). Second, the proxy acts on the incoming queries with $ttl > 0$ by directly replying to them if the corresponding query hits are already cached. It then passes the remaining $Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}$ queries to servent A, where f_{miss_rvs} is the query miss ratio in reverse caching. Servent A then forwards these queries to each of its remaining $(C - 1)$ neighbors in the p2p network.

Table 1 summarizes the number of queries and query hits sent along each of the four directions. Query hits travel in opposite directions to their query counterparts. In forward caching (Figure 2(a)), $Q \cdot f_{ttl>0} \cdot f_{miss\_fwd} \cdot (C-1)$ queries sent to outside the gateway result in $Q \cdot f_{miss\_fwd} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$ query hits, where $C_{QH}(k)$ is the average number of query hits for a query with $ttl = k$. Combined with the query hits from cache hits in the forward cache, the total number of query hits sent by the proxy to servent A is $Q \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$. Finally, $Q \cdot L_{QH}$ query hits are generated from servent A's local object store, and sent back to servent $A1$, along with the query hits received by A from the proxy.

The query hits in reverse caching (Figure 2(b)) are also shown in Table 1. Out of the Q queries received from $A1$, $Q \cdot f_{ttl=0} + Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}$ are sent to A by the proxy, as it does not cache query hits for queries with $ttl = 0$. A in turn forwards the $Q \cdot f_{ttl>0} \cdot f_{miss\_rvs} \cdot (C-1)$ duplicated queries to outside servents, and receives back $Q \cdot f_{miss\_rvs} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$ query hits. Servent A then sends these query hits along with $(Q \cdot f_{ttl=0} + Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}) \cdot L_{QH}$ query hits generated locally to the proxy. Finally, the proxy sends all the query hits forwarded from A along with $Q \cdot f_{hit\_rvs} \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k+1)$ query hits from its cache to $A1$.

The following conclusions can be drawn from Table 1.

– The volumes of the queries sent to outside in forward caching and in reverse caching ((4) in Table 1) would be the same if f_{miss_fwd} and f_{miss_rvs} are equal.
– The volumes of the query hits received from outside in forward caching and in reverse caching ((5) in Table 1) would be the same if f_{miss_fwd} and f_{miss_rvs} are equal.
– The total number of query hits sent back to outside (e.g., to servent $A1$) in forward caching and in reverse caching ((8) in Table 1) are the same. This makes sense as caching is expected to preserve the end user experience in searching data files.

Thus the effectiveness comparison of forward and reverse caching boils down to the relative values of the query cache miss ratios in forward and reverse caching, i.e., f_{miss_fwd} and f_{miss_rvs}.

Assuming the forward and reverse caches have the same capacity, i.e., they can store the same total number of query hits for however many queries, the cache hit

Table 1. Query and query hit traffic going in and out hijacked connections of servent A, triggered by Q queries coming in from one neighbor servent outside the gateway. C is the number of neighbors in the p2p overlay. All neighbors are outside the gateway. L_{QH} is the average query hits per query from servent A's local object store. $C_{QH}(k)$ is the average number of query hits for a query with $ttl = k$. $f_{ttl=k}$ denotes the fractions of all Q queries with $ttl = k$ after decrementing at A. The derivation in (8) for reverse caching uses $C_{QH}(k+1) = (C-1) \cdot C_{QH}(k) + L_{QH}$.

Category	Forward Caching
Queries:	
(1) rcvd from outside	Q
(2) sent to inside	Q
(3) rcvd from inside	$Q \cdot f_{ttl>0} \cdot (C-1)$
(4) sent to outside	$Q \cdot f_{ttl>0} \cdot f_{miss\_fwd} \cdot (C-1)$
Query Hits:	
(5) rcvd from outside	$Q \cdot f_{miss\_fwd} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$
(6) sent to inside	$Q \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$
(7) recv from inside	$Q \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k) + Q \cdot L_{QH}$
(8) sent to outside	$Q \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k) + Q \cdot L_{QH}$

Category	Reverse Caching
Queries:	
(1) rcvd from outside	Q
(2) sent to inside	$Q \cdot f_{ttl=0} + Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}$
(3) rcvd from inside	$Q \cdot f_{ttl>0} \cdot f_{miss\_rvs} \cdot (C-1)$
(4) sent to outside	$Q \cdot f_{ttl>0} \cdot f_{miss\_rvs} \cdot (C-1)$
Query Hits:	
(5) rcvd from outside	$Q \cdot f_{miss\_rvs} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$
(6) sent to inside	$Q \cdot f_{miss\_rvs} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$
(7) recv from inside	$Q \cdot f_{miss\_rvs} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$ $+(Q \cdot f_{ttl=0} + Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}) \cdot L_{QH}$
(8) sent to outside	$Q \cdot f_{miss\_rvs} \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k)$ $+(Q \cdot f_{ttl=0} + Q \cdot f_{ttl>0} \cdot f_{miss\_rvs}) \cdot L_{QH}$ $+Q \cdot f_{hit\_rvs} \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k+1)$ $= Q \cdot (C-1) \cdot \sum_{k=1}^{6} f_{ttl=k} \cdot C_{QH}(k) + Q \cdot L_{QH}$

ratios in forward and reverse caching should be similar for the following reasons. First of all, the localities in the queries filtered by the forward cache and the reverse cache are the same. This is because the $Q \cdot f_{ttl>0} \cdot (C-1)$ queries acted on by the forward cache is from duplicating $(C-1)$ times the $Q \cdot f_{ttl>0}$ queries that would be seen by the reverse cache, one for each of A's $(C-1)$ neighbors outside the gateway, and the caching algorithm distinguishes the same query forwarded to different next hop nodes.

Moreover, since the number of query hits to be received by the proxy from sending $Q \cdot f_{ttl>0}$ queries to servent A in reverse caching is similar to the number of query hits to be received by the proxy from sending the duplicated $Q \cdot f_{ttl>0} \cdot (C-1)$ queries to $A2$ and $A3$ (after decrementing the TTL), assuming the difference – the query hits generated by A locally – is insignificant, the total number of query hits received by the proxy is expected to be similar in forward and reverse caching. Thus if the proxy cache has the same capacity in terms of the number of query hits, and the queries have the same locality, the hit ratios are expected to be comparable.

Finally, consider the case where query hits expire before the cache capacity is reached. Since the cache hits stored in the reverse proxy and in the forward proxy effectively correspond to the same set of queries, i.e., those whose query hits have not expired, the hit ratios in the two caches are again expected to be comparable.

4 Experiments

We implemented both forward and reverse query caching proxies. In the following, we experimentally compare the two proxies in a testbed consisting of eight machines running eight Gnutella servents.

4.1 Experimental Setup

Our testbed consists of a cluster of eight PCs running FreeBSD 4.6, each of which runs a Gnutella servent, configured to allow zero incoming and three outgoing connections. Each servent is passive; it only forwards queries and query hits, but does not initiate any queries. Furthermore, it does not store any files for sharing.

To simplify the setup, instead of using a real router, we configured each servent machine to use the caching proxy machine as the default router. IP forwarding rules are specified on the caching proxy machine such that packets going to port 6346 of any destination will be forwarded to port 6346 of localhost, and all other traffic are forwarded. Thus only outgoing Gnutella connections will be hijacked by the proxy.

We started the experiments with all eight servents at 5:00am EST on May 8, 2004 (after a 30-minute warm-up period), and the experiment lasted for an hour. In all tests, we find the neighboring servents of the eight servents behind the caching proxies to be distinct. The proxy recorded all Gnutella packets going in and out on the hijacked outgoing connections.

4.2 Results

We fixed the total number of query hits the cache can store to be 350000 and compared the caching results under forward and reverse caching. The cache replacement policy was LRU. In addition, a 30-minute expiration is imposed for cached query hits. However, in our experiments, the cache capacity was reached first before any query

Table 2. Query and query hit traffic going in and out the cache. The cache stores up to a total of 350000 query hits for different queries.

Category	Forward Caching	Reverse Caching
Time measured (EST)	6:30-7:30am May 8, 2004	5:00-6:00am May 8, 2004
Average # connections/servent	2.70	2.69
Cache Miss Ratio	58.03%	56.07%
Queries:		
rcvd from outside	1503770	1531813
sent to inside	1503770	1220787
with $ttl > 1$	656804	654221
rcvd from inside	1102104	578992
sent to outside	604042	578992
queries dropped at cache	55656	42134
Query Hits:		
rcvd from outside	44356	42814
sent to inside	75938	42814
recv from inside	75938	42814
sent to outside	75938	80175
Average query hit size/query (Bytes)	4693.16	4857.32

hit expired. The measured statistical results of the two caching proxies are shown in Table 2. Note while the threshold for triggering cache replacement is the total number of cached query hits, the granularity for replacement is a query entry, i.e., all of its query hits.

For the forward cache, 1503770 queries crossed the cache and were forwarded to the servents. Out of these, only 656804 have $ttl > 1$ (or $ttl > 0$ after decrementing) and would be forwarded to the remote servents outside the gateway by the servents inside. The average number of connections per servent was 2.70, averaged over the duration of the experiment. Hence, the total number of queries received by the proxy from inside would be about $656804 * 1.70 = 1116567$. The observed value of 1102104 was within 1.3% of this. Only 58.03% (miss ratio) of these would be sent to outside. The actual number of queries sent outside was slightly lower than this number due to dropped queries by the proxy because of repeated `muid` (Section 2.3).

The number of query hits received from outside was 44356, an additional 31582 were serviced from the cache, for a total of 75938. Notice that the percentage of traffic served from cache, 41.6%, is the saving in bandwidth that the forward cache provides, assuming a constant number of bytes per query hit.

For the reverse cache, 1531813 queries were received from outside of which 654221 have $ttl > 1$. The reverse proxy performed caching on them and the cache miss ratio was 56.07%. The proxy then sent the 366822 misses along with the 877592 queries with $ttl = 1$ to the servents inside, and thus the total number queries sent inside was expected to be 1244414. Due to queries dropped by the cache (becasue of repeated `muid`), the actual number of queries sent inside (1220787) was about 1.9% lower. The average number of connections per servent was 2.69, averaged over the duration of the experiment. Thus the expected number of queries received from the servents inside would be $366822 * 1.69 = 619929$. Again, due to dropped queries by the cache when sending inside, fewer than 366822 queries were sent inside, and the actual number received from inside was 578992.

In terms of query hits, a total of 80175 replies to outside servents were sent out of which 42814 were received from outside, and the rest were served out of the cache. Thus the bandwidth saving for query hits in the reverse cache is 46.6%, assuming a constant number of bytes per query hit.

Comparing forward caching and reverse caching, the cache miss ratios, the numbers of queries sent to outside, and the numbers of query hits received from outside are within 2.0%, 4.1%, 3.5%, respectively. The total numbers of query hits sent outside for forward and reverse caching are within 0.63% of each other. This confirms the analysis that if the cache capacity is in terms of the number of cached query hits, the traffic reduction will be comparable in reverse and forward caching.

5 Related Work

There have been many studies that measured, modeled, or analyzed peer-to-peer file sharing systems such as Gnutella (for example, [6, 7]) and Kazaa (for example, [8, 9]). Many of these studies also discussed the potential of caching data object files (for example, [8]) or retrieving files from other peers within the same organization [9] in reducing the bandwidth consumption.

Several previous work studied query caching in Gnutella networks. Sripanidkulchai [1] observed that the popularity of query strings follows a Zipf-like distribution, and proposed and evaluated a simple query caching scheme by modifying a Gnutella servent. The caching scheme proposed was fairly simple; it caches query hits solely based on their query strings and ignores TTL values. In [2], Markatos studied one hour of Gnutella traffic traces collected at three servents located in Greece, Norway, and USA, and proposed a query caching scheme by modifying servents to cache query hits according to the query string, the forwarder from which the query is forwarded, and the TTL. In our previous work [3], we proposed transparent forward query caching at the gateway and experimentally showed its effectiveness. This paper builds on top of our previous work and shows that reverse and forward query caching are equally effective in the context of transparent query caching at the gateway.

Several recent work studied other p2p traffic. Leibowitz et al. [10] studied one month of FastTrack-based [11] p2p traffic at a major ISP and found that the majority of p2p files are audio files and the majority of the traffic are due to video and application files. They also reported significant locality in the studied p2p data files. Saroiu et al. [12] studied the breakdowns of Internet traffic going through the gateway of a large organization into web, CDN, and p2p (Gnutella and Kazaa) traffic. They focused on HTTP traffic. In contrast, this paper focuses on the p2p protocol traffic, and compares different transparent caching schemes for query traffic.

6 Conclusions

In this paper, we studied the effectiveness of forward and reverse caching of p2p query traffic at the gateway of an organization or ISP via analysis and experimental measurement. Our study showed that forward caching and reverse caching at the gateway are equally effective in reducing query and query reply traffic across the gateway. Since in a peer-to-peer network the communication are symmetric – query and query hit traffic travel on both incoming and outgoing connections (with respect to the peers inside the gateway), transparent caching will be even more effective if traffic on both type of connections are filtered.

Acknowledgment

This work was supported in part by Cisco Systems through the University Research Program and by Purdue Research Foundation.

References

1. Sripanidkulchai, K.: The popularity of gnutella queries and its implication on scaling. ⟨ http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html ⟩ (2001)
2. Markatos, E.P.: Tracing a large-scale peer to peer system: an hour in the life of gnutella. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'02). (2002)
3. Patro, S., Hu, Y.C.: Transparent Query Caching in Peer-to-Peer Overlay Networks. In: Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03). (2003)
4. Clip2: The Gnutella protocol specification. ⟨ http://dss.clip2.com/GnutellaProtocol04.pdf ⟩ (2000)
5. Kirk, P.: The Gnutella 0.6 protocol draft. ⟨ http://rfc-gnutella.sourceforge.net/ ⟩ (2003)
6. Adar, E., Huberman, B.: Free riding on gnutella. First Monday **5** (2000)
7. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking (MMCN'02). (2002)
8. Saroiu, S., Gummadi, K.P., Dunn, R.J., Gribble, S.D., Levy, H.M.: An analysis of internet content delivery systems. In: Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI'02). (2002)
9. Gummadi, K.P., Dunn, R.J., Saroiu, S., Gribble, S.D., Levy, H.M., Zahorjan, J.: Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP'03). (2003)
10. Leibowitz, N., Bergman, A., Ben-Shaul, R., Shavit, A.: Are file swapping networks cacheable? Characterizing p2p traffic. In: Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW7). (2002)
11. Truelove, K., Chasin, A.: Morpheus out of the underworld. The O'Rielly Network, ⟨ http://www.openp2p.com/pub/a/p2p/2001/07/02/morpheus.html ⟩ (2001)
12. Saroiu, S., Gummadi, P.K., Dunn, R.J., Gribble, S.D., , Levy, H.M.: An analysis of internet content delivery systems. In: Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI'02). (2002)