Design and Implementation of A P2P Cooperative Proxy Cache System

James Z. Wang Vipul Bhulawala
Department of Computer Science
Clemson University, Box 340974
Clemson, SC 29634-0974, USA
+1-864-656-7678
{jzwang, vbhulaw}@cs.clemson.edu

Abstract

In this paper, we design and implement a P2P cooperative proxy caching system based on a novel P2P cooperative proxy caching scheme. To effectively locate the cached web documents, a TTL-based routing protocol is proposed to manage the query and response messages in the P2P cooperative proxy cache system. Furthermore, we design a predict query-route algorithm to improve the TTL-based routing protocol by adding extra information in the query message packets. Our performance studies demonstrate that the proposed message routing protocols significantly improve the performance of the P2P cooperative proxy cache system, in terms of cache hit ratio, byte hit ratio, user request latency, and the number of query messages generated in the proxy cache system, compared to the flooding based message routing protocol.

1. Introduction

Several cooperative proxy cache systems, such as harvest [1] and its successor Squid [2], have been developed in recent years and are available in open source format. However these cooperative proxy cache systems have their limitations [3, 4]. To deal with the drawbacks in these cooperative proxy cache systems, we proposed a novel P2P cooperative proxy caching scheme [5] based on an individual-based caching model. In this paper, we design and implement a P2P cooperative proxy cache system based on the proposed P2P cooperative proxy caching scheme. Specifically, we propose and evaluate different message routing protocols for data search, data cache and data replication. The goal is to implement a P2P cooperative proxy cache system that can increase cache hit ratio and reduce user request latencies with the least complexity.

The remainder of this paper is organized as follows. In section 2, we briefly discuss the P2P cooperative proxy caching scheme to make the paper self-contained. In section 3, we describe the design and architecture of our proxy server. We propose the TTL-based message routing protocol and the predict query-route (PQR) algorithm in section 4. In section 5, we use experimental study to evaluate our proposed message routing protocols. We give our conclusion and discuss future studies in section 6.

2. P2P proxy caching scheme

Unlike other existing proxy caching schemes in which the proxy cache systems are manually configured into certain hierarchies based on underlying network structures, all individual proxies in our cooperative proxy cache system automatically discover their neighbors and virtually link the cooperative proxies into a Virtual Proxy Graph (VPG), which is an overlay network independent of underlying network topology. With a VPG, the aggregate effect of data movement among the proxy neighbors creates a life-like group behavior that automatically distributes the data closer to clients with less complexity [5].

To facilitate the data search, data cache and data replication based on the VPG, we proposed a network cache line for cached web documents. Figure 1 and Figure 2 demonstrate a simple VPG and a network cache line respectively. A network cache line is made up of two portions. The body portion is used to store the web document itself. The head portion consists of ID, Tag, state bytes and link fields. The ID field contains a key which the URL of the cached web document will be hashed to. The Tag is the name of the document. State bytes are used to store information for cache coherency as well as some other statistical data, such as number of replicas of the web document known to this node in the proxy cache system and shortest distance to the nearby replicas. A link field is organized as a pair of integer <NID, Dist> to provide the link information. NID is an integer used to lookup the neighbor table to get the neighbor's IP address. Dist is round trip distance in finding the cached web document through the neighbor specified by NID.

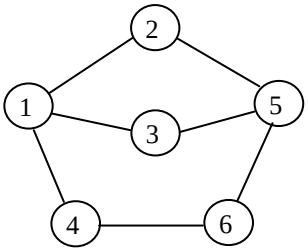

Figure 1: Virtual proxy graph.

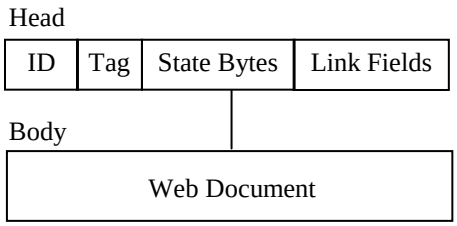

Figure 2: A network cache line.

An essential part of the proposed proxy caching scheme is the individual-based data search and data cache strategies with which individual proxies can manage the cached data on the proposed *VPG* using the designed network cache line based on their local knowledge of the global cache state. The data search, data cache and data replication actions at individual proxies create a *life-like* group behavior so that the aggregate effect of the individual caching actions can maintain a globally optimized caching state for the cooperative proxy cache system [5]. When a new web request reaches a certain proxy server from its client, the proxy server takes actions based on three different situations as follows:

1. ***The entire network cache line is cached in this proxy server:*** In this case, the cached web document is sent to the client and the web request is satisfied.
2. ***Only head of the network cache line is cached:*** The current proxy checks the head of the network cache line for the shortest distance to the cached web replicas. If the distance to the nearest web replica is longer than its expected response time by directly requesting from the original web server, a query is sent to the original web server. Otherwise, a query message will be created, and propagated along the links in the heads of network cache lines to the proxy server that holds the nearest web replica.
3. ***Nothing is cached at this proxy server:*** The proxy server first sends a query message to all neighbor proxies and waits for the responses from them. The query message will be disseminated to other proxies through neighbor relay until the message is expired or a matching network cache line (maybe head only) is reached. The proxy that has the matching network cache line responds the query. The responding message is routed back to the requesting proxy along the query path. The proxies on the routing path create or update the appropriate network cache line head based on the responding message to provide links to the cached web replica. Once the responding messages reach the requesting proxy and related network cache line head is created on that proxy, the rest of search process would be same as in case 2.

When a query locates a cached web document in the proxy cache system, the document will be transferred to the requesting proxy from the proxy server where it is located.

3. Design and implementation of the proxy server

The proxy servers designed based on our P2P proxy caching scheme need to not only cache the content from the web servers but also cooperate with their neighbor proxies in the cooperative proxy cache system. The basic components of a proxy server are demonstrated in Figure 3.

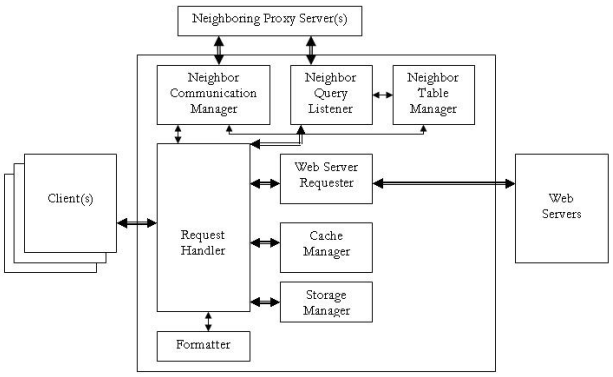

Figure 3: Proxy server components

3.1 Proxy server components

A proxy server consists of eight major components, including *formatter, request handler, storage manager, cache manager, web server requester, neighbor communication manager, neighbor query listener, and neighbor table manager.* The functions of these components are described as follows.

Formatter: This component accepts and validates the raw URL as requested by the client browser, and then converts the URL string into a standard format:

http://<url-body>:<port-number>/<path of the URL>

Request Handler: The request handler takes the web request from the client (a web browser) and passes it to the formatter. Upon receiving the formatted URL, it checks with the cache manager for a local cached copy. If a copy is found, it asks the storage manager to retrieve the document from the disk and send it to the client. If no

cached copy is found in this proxy cache, the web request is passed to the neighbor communication manager to send it to the neighbor proxies. If a neighbor proxy returns a cache hit, the web document is retrieved from that proxy.

Cache Manager: The cache manager is used to manage the cache space. It has a hash table for storing the URLs of the client requests. Given a URL string ***s*** with length n, we can represent it by a character array ***s***[0],..,***s***[n] with ***s***[n] equal to the null character. Then the hash algorithm for the URL string ***s*** is

$$h = s[0]\times 31^{n-1} + s[1]\times 31^{n-2} + \cdots + s[n-1]$$

We apply the modulus operator, denoted by %, with the number of elements in the table to obtain a valid array index, so that the data is inserted directly into the array. To look up for that item the search key is hashed to pick the right element in the array. Cache manager also handles different replacement policies implemented in the proxy server and updates the corresponding data structures.

Storage Manager: The storage manager is used to manage the actual cached web documents on the disk. It uses hashing function to distribute large number of files in different directories rather than to store them in one single directory. The storage manager performs basic storing, retrieving and updating functions on the cached web documents.

Web Server Requester: This module sends HTTP requests to the original web servers and retrieves the web documents back to the local cache. If the disk cache space is not enough, it calls the cache manager to free enough disk cache space before downloading the requested web document to the disk.

Neighbor Communication Manager: Each proxy server needs to communicate with its neighbor proxies to look for documents cached in their cache space. This module prepares query messages and sends them to appropriate neighbor proxies. After that, it waits for responses. It maintains a timer to monitor the message waiting time. When waiting time expires, it returns to the request handler with a list of available responses. It is also responsible for maintaining the neighbor table and the most recent query sequence number of each neighbor proxy.

Neighbor Query Listener: This is a standalone thread which listens for neighbor proxy's queries. It parses the query message and extracts the URL from the message, then sends it to the cache manager to find out whether a cached web document exists. If a cached copy is found, a response is sent back to the requesting neighbor proxy with the information about the cached web document. If no cached copy is found, it either forwards the query to other neighbors, if needed, or sends back a response stating that the document was not found.

Neighbor Table Manager: The neighbor table manager maintains information about the neighbor proxy servers including IP addresses, port numbers, proxy server state, and some other routing related information.

Besides these eight essential components, we also use a thread-pool manager to manage a set of concurrent communication channels. This module re-uses threads instead of spawning new threads every time a thread is needed for a client or neighbor node connection.

3.2 Implementation

The proxy server is developed under Linux and Solaris systems using C programming language. Its main function call loop is depicted in Figure 4.

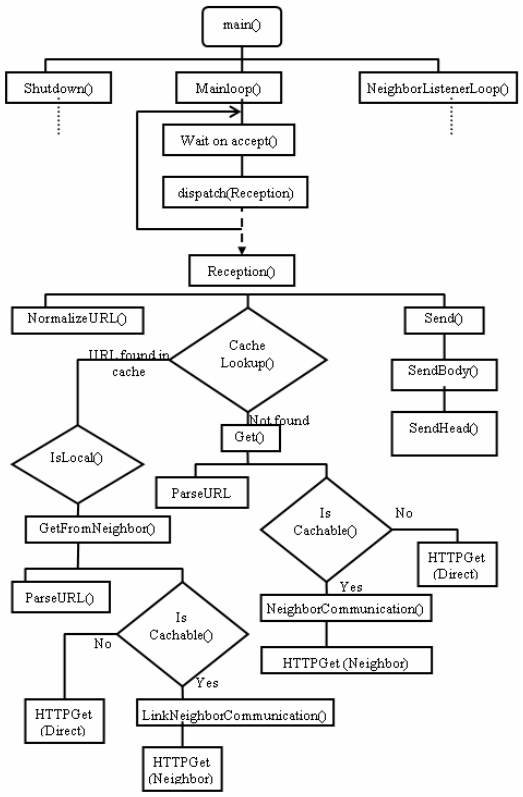

Figure 4: Main function call graph.

Individual caching proxies cooperating with other proxy servers for data search, data cache and date replication is the most important feature of our cooperative proxy server. This is demonstrated by neighbor listener call graph shown in Figure 5. The proxy server listens on a specified port for incoming HTTP requests from its clients. For each incoming web request, it dispatches a thread from the thread-pool to serve the particular web request. Since it is designed as a multi-threaded server, it can handle multiple web requests simultaneously. The proxy server also listens on a different port for queries from its neighbor proxies. It maintains a separate thread-pool for handling these queries from neighbor proxies.

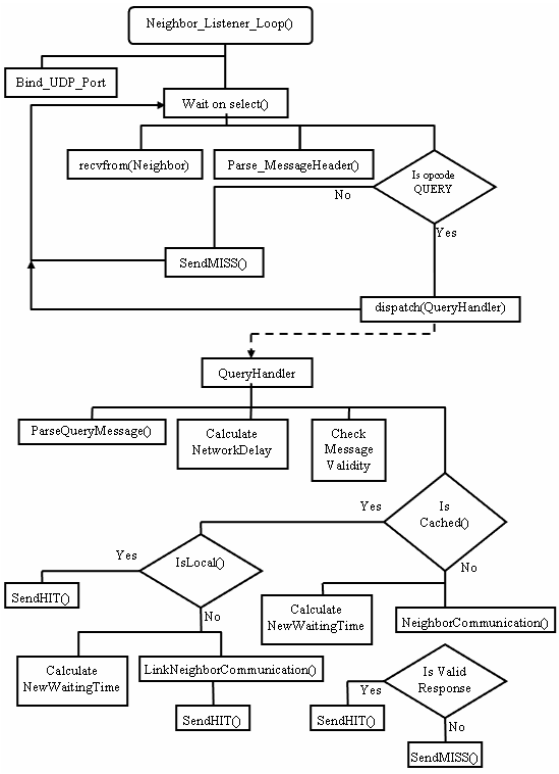

Figure 5: Neighbor listener call graph

4. Message routing protocol

Message routing protocols are used to specifically handle the inter-proxy requests for our P2P cooperative proxy cache system. To support the P2P cooperative proxy caching scheme, our message routing protocol should not only carry query and response information but also carry document header information. We design a lightweight message format on top of User Datagram Protocol (UDP).

4.1 TTL-based routing protocol

In TTL-based routing protocol, each message contains a message header followed by the message body. Message header contains information describing the message type, message length, the source proxy server and the unique identifier of the message. Detailed descriptions of the fields in message header are presented in Table 1. Different types of messages have different sizes. Query messages have the biggest payload. A query message contains timestamp of the query issuing time, the query expiration time, TTL, the list of proxy nodes the message was routed through, and the URL of the web document. The functionalities of these fields are explained in Table 2. The Hit message is sent in response of a query message when a proxy node finds a match for the query URL in its cache. It contains RTT, port number, IP address and URL. The detailed definitions of these fields are listed in Table 3. If the requested web document is not in current proxy cache, a Miss message is used to notify the sender that no information was found for the requested URL. A Miss message only contains the URL as the payload.

Table 1: Message header

Opcode	Each message has a unique opcode to specify the message type.
	QUERY: 1 HIT: 2 MISS: 3
Length	This field is used to determine the length of the URL in the payload.
Request Number	A unique request number is allocated to every Query message. All the responses (HITs and MISSes) carry the same request number to identify it as a response to that particular request.
Neighbor ID	It represents the neighbor ID from whom the proxy received the message.

Table 2: Query message body

Timestamp	Timestamp to calculate network latency on the receiver end.
Query Expiration time	Maximum amount of time the original node would wait for the responses from its neighbors. As queries are forwarded along the path, this time reducing by the sum of network delay and service times in the path of the query.
Number of nodes passed /TTL	The number of nodes passed by the Query. It also serves as a TTL field. If the number of passed nodes reaches a predefined maximum value the query is no longer forwarded.
URL	URL of the requested web document. It also hashes to the ID field of the web cache line.
Passed Nodes	A list of nodes stored in a Query message which represents nodes through which the message was propagated.

Table 3: Hit message body

RTT	Round Trip Distance to the document. It gets incremented by the network delay and service times of each node in the path.
Port	Port number of the destination proxy node.
IP Address	IP Address of the destination proxy node
URL	URL of the requested web document.

Since our individual-based P2P proxy caching scheme requires that each caching proxy communicates only with its neighbors. Our TTL-based message routing protocol keeps track of the nodes that the query message has passed through. Each node adds its ID to the query message when it sends the message to its neighbors. A proxy node will discard a received query message if either its TTL reaches the predefined maximum value, or it finds that its own ID is included in the query message it just received.

4.2 Predict query-route algorithm

The basic TTL-based message routing protocol may generate too many unnecessary messages in the P2P cooperative proxy cache system. To reduce the number of redundant messages, we propose a new message routing algorithm, predict query-route (PQR) algorithm, which records the forward routes of the query message in addition to the sender's ID. In this algorithm, before forwarding the query message to its neighbor proxies, each proxy node adds its own ID and the IDs of the neighbor proxies into the message body. To demonstrate the advantage of the predict query-route algorithm, we use the VPG shown in Figure 1 as an example to study the number of query messages generated using different message routing protocols in the proxy cache system.

We assume a client in proxy node 1 requests a web document that is not currently cached in the cooperative proxy cache system. A query message is generated in node 1 as it does not find any information about the web document in its local cache. Table 4 shows query messages generated in the cooperative cache system using the TTL-based routing protocol with TTL = 4.

Table 4: Messages using TTL-based routing protocol

Routing Steps	Useful Query (Q) and Redundant Message (R)	Query Received by node #
Step 1	Q{1}	2
	Q{1}	3
	Q{1}	4
Step 2	Q{1,2}, R{1,3}	5
	Q{1,4}	6
Step 3	R{1,3,5}	2
	R{1,2,5}	3
	R{1,4,6}	5
	R{1,2,5} R{1,3,5}	6
Step 4	R{1,2,5,6} R{1,3,5,6}	4

In Table 4, each message is simply presented by the set of proxy node IDs added to the message body. For instance, the query message sent by node 1 is presented as Q{1}. When node 2 receives message Q{1} and forwards it to node 5, the message is presented as Q{1, 2}. If a query message is forwarded to a node that the same query has already been propagated to, we use R to represent the message. For example, assume node 5 has already received the same query from node 2 when query message from node 3 arrives at node 5. Then the query message from node 3 becomes redundant. We use R{1, 3} to represent this redundant query message. As shown in Table 4, 8 out of 13 messages are redundant using the TTL-based message routing protocol.

Table 5: Messages using predict query-route algorithm

Routing Steps	Useful Query (Q) and Redundant Message (R)	Query Received by node #
Step 1	Q{**1**,2,3,4}	2
	Q{**1**,3,4,2}	3
	Q{**1**,4,2,3}	4
Step 2	Q{1,**2**,3,4,5}, R{1,**3**,2,4,5}	5
	Q{1,**4**,2,3,6}	6
Step 3	R{1,4,2,3,**6**,5}	5
	R{1,2,3,4,**5**,6}, R{1,3,2,4,**5**,6}	6

Now we use predict query-route algorithm on the same scenario. When node 1 sends query message to its neighbor node 2, 3, and 4. Besides adding the ID of node 1 in the message body, the IDs of node 2, 3, and 4 are also added in the message body. Table 5 lists all messages generated in the cooperative proxy cache system using the predict query-route algorithm. The messages sent to node 2, 3 and 4 by node 1 are presented as Q{**1**,2,3,4}, Q{**1**,3,4,2} and Q{**1**,4,2,3} respectively. Similarly the message sent to node 5 from node 3 is redundant since node 5 has already received the same query from node 2. This redundant message is shown as R{1,**3**,2,4,5} in Table 5.

Using the predict query-route algorithm, the query message propagation stops after 3 steps. On the other hand, it takes 4 steps for the proxy cache system to stop forwarding messages using the TTL-based message routing protocol. In this particular example, the predict query-route algorithm generates 50% less redundant messages in the cooperative proxy cache system.

5. Experimental study

To evaluate our proposed message routing protocols, we compare the TTL-based message routing protocol and the predict query-route algorithm with the flooding based routing algorithm using experimental studies. The latest web traces obtained from irCache [6] were used to evaluate the message routing protocols and algorithms. The web trace log contains 200,000 web requests. Due to the length of web traces, the experimental studies last for months.

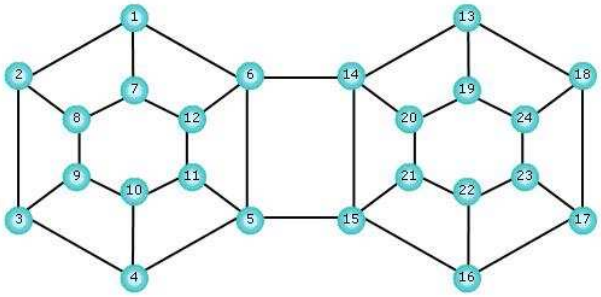

Figure 6: VPG used for experimental studies.

Our cooperative proxy cache system was set up with 24 proxy servers. To guarantee the consistency of the performance study, we manually configure a VPG, as shown in Figure 6, so that each proxy connects to the same set of neighbor proxies in all experimental studies. The VPG is in the form of two spider webs connected by two links through four nodes. Each spider web is in a subnet of the network. For those nodes in the same subnet, it makes sense for each node to have the same number of neighbors, because all nodes are close to each other in the network with sufficient cache space, network bandwidth, and computing power. We choose TTL = 4 for the message routing protocols. The message expiration time is set to 2 seconds. Those settings are based on the structure of VPG and web response data collected by running the benchmark Polygraph [7] on a web caching appliance developed by Swell Technology [8].

We developed an automatic web request tool based on Webjamma, an open source stress test tool for HTTP proxy server maintained by Tom Johnson [9]. Our web request tool sequentially scans the set of URLs from the web log trace file and randomly sends GET requests to one of the 24 proxy servers. Each proxy server in the cooperative proxy cache system is a Sun workstation with a sparcv9 processor operating at 650 MHz and 512 MB memory.

We use cache hit ratio, byte hit ratio, average user request latencies and the average number of messages sent by each proxy node as our system performance metrics. Cache hit ratio is the percentage of total web requests satisfied by the proxy cache system. Byte hit ratio is the percentage of web data sent to the clients by the proxy cache system. A total of 200,000 web requests are issued by the web request tool. The experimental studies lasted for two months during off-peak hours to avoid disrupting the normal network traffic and to avoid other network traffic causing unexpected delays in our message routing.

The links among the cached web documents established by our P2P cooperative proxy caching scheme can reduce the number of messages needed for searching the cached web documents in the P2P cooperative proxy cache system. Our TTL-based message routing protocol is designed to take advantage of the link information to provide better system performance. Once the query message reaches a proxy server that contains link information for the requested web document, the message will be forwarded along the linked path in stead of propagating to all neighbor proxies. We compare performance of the proxy cache system using our proposed message routing protocols with that using a flooding based message routing protocol.

The flooding based message routing protocol is similar to our TTL-based message routing protocol except no link information and no message expiration time are used in the protocol. Whenever there is a local cache miss for a web request, the caching proxy would have to broadcast a query message for the web document to its neighbors. The caching proxies propagate the query message until the TTL of the message reaches a predefined maximum value. As in our TTL-based message routing protocol, the maximum TTL value for flooding message routing protocol is set to 4. The LRU algorithm is used for cache replacement. The experimental results are depicted in Figure 7, Figure 8, Figure 9 and Figure 10.

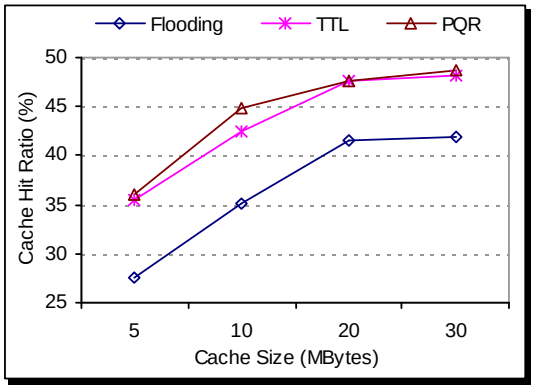

Figure 7: Cache hit ratio under different message routing protocols.

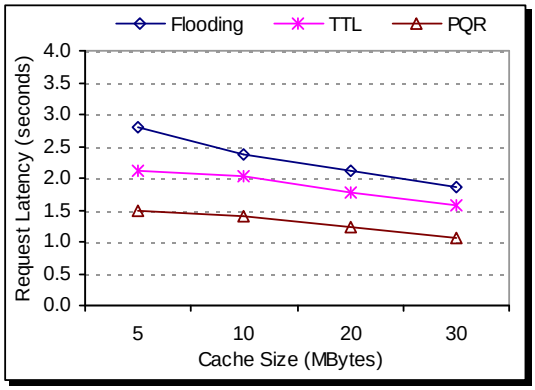

Figure 8: Average user request latencies under different message routing protocols.

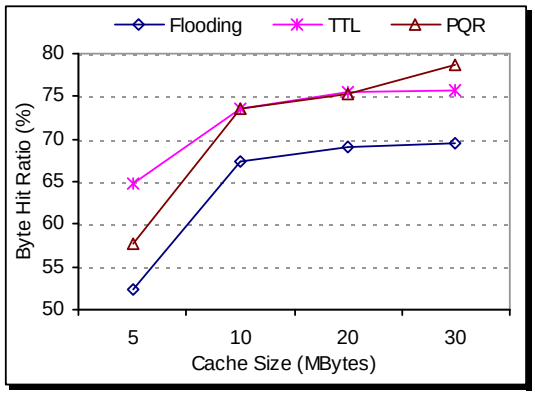

Figure 9: Byte hit ratio under different message routing protocols.

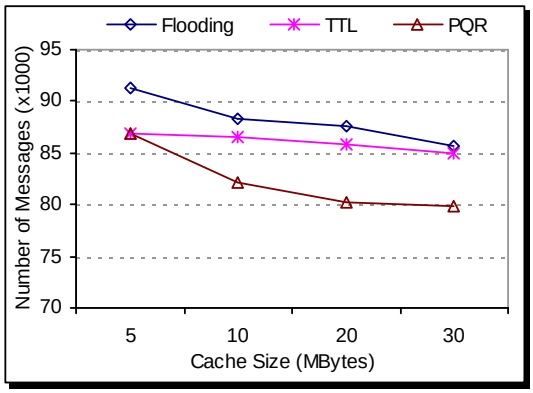

Figure 10: Average number of messages sent by each proxy under different message

As demonstrated in Figure 7 and Figure 9, the proxy cache system using our proposed message routing protocols result in higher cache hit ratio and byte hit ratio. These achievements are primarily due to our P2P cooperative proxy caching scheme which can link cached web documents through replicated network cache lines. Since more web requests can be satisfied by the proxy cache system using our proposed message routing protocols, the average user request latencies are reduced as shown in Figure 8. The predict query-route protocol shows clearly better performance than the TTL-based message routing protocol in terms of user request latency, although they both have comparable cache hit ratio and byte hit ratio. The results demonstrate that both message routing protocols have equal probability of finding the cached web documents in the cooperative proxy cache system. However the predict query routing protocol can discover the cached web documents more quickly, which is consistent with the results of our example demonstrated in section 4.2.

Another purpose of the predict query-route algorithm is to reduce the average number of messages sent by each proxy node. As proven in Figure 10, the predict query-route algorithm generally outperforms the TTL-based message routing protocol in terms of the number of messages sent by each proxy node. This is also in agreement with our analysis in section 4.2. Yet the TTL-based message routing protocol shows only marginal improvement over the flooding based approach in terms of the average number of messages sent by each proxy node.

6. Conclusion and future studies

In this paper, we design and implement a P2P cooperative proxy cache system based on a novel P2P cooperative proxy caching scheme. To efficiently locate the cached web documents in the cooperative proxy cache system, various message routing protocols are designed for proxy servers to exchange messages. In particular, a TTL-based routing protocol is proposed to manage the query and response messages in the proxy cache system. Furthermore, a predict query-route (PQR) algorithm is proposed to improve the TTL-based routing protocol by adding extra information in the query packets, thus reducing the number of redundant messages in the cooperative proxy cache system. Our experimental results show that our TTL-based message routing protocol and the predict query-route algorithm can take advantage of the unique features of the P2P cooperative proxy caching scheme, and significantly improve the performance of the cooperative proxy cache system, compared to the flooding based message routing protocol.

Currently we are improving our cooperative proxy cache system to support FTP and SSL. We will also study the impact of cache coherency to the system performance, so that an appropriate cache coherency policy can be integrated into the cooperative proxy cache system. A simple LRU algorithm was used as the cache replacement policy in our performance study. We will further investigate various factors that affect the cache replacement and find the best cache replacement algorithm for our P2P cooperative proxy cache system.

7. References

[1] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In Computer Networks and ISDN Systems,Dec. 1995.

[2] Squid Web Proxy Cache, www.squid-cache.org/

[3] M. Rabinovich, J. Chase and S. Gadde. Not All Hits Are Created Equal: Cooperative Proxy Cache Over a Wide-Area Network. Computer Networks and ISDN Systems, 30(22-23):2253-2259, November 1998.

[4] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. Proc. of l7th ACM Symposium on Operating Systems Principles (SOSP), 1999.

[5] James Z. Wang and Ratan K. Guha. “Proxy Ecology - Cooperative Proxies with Artificial Life.” In Proceedings of IEEE/WIC IAT-2003, Halifax, Canada. 2003.

[6] IRCache project, http://www.ircache.net/

[7] Web PolyGraph, http://www.web-polygraph.org/

[8] Swell Technology, http://www.swelltech.com/products/caching.html

[9] WWW Traffic Analysis Tools, http://research.cs.vt.edu/nrg/webjamma.html